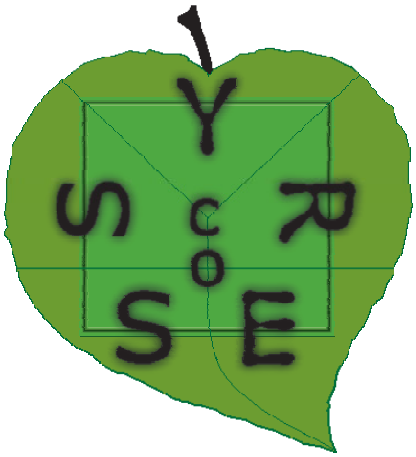


SYRCoSE 2012

Editors:

Alexander Kamkin, Alexander Petrenko,
Andrey Terekhov

Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering

Perm, May 30-31, 2012

Perm
2012

Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012), May 30-31, 2012 – Perm, Russia:

The issue contains the papers presented at the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012) held in Perm, Russia on 30th and 31st of May, 2012. Paper selection was based on a competitive peer review process being done by the program committee. Both regular and research-in-progress papers were considered acceptable for the colloquium.

The topics of the colloquium include software development technologies, programming languages, testing and verification of computer systems, analysis of programs, information systems, image and video processing and others.

Труды 6-ого весеннего/летнего коллоквиума молодых исследователей в области программной инженерии (SYRCoSE 2012), 30-31 мая 2012 г. – Пермь, Россия:

Сборник содержит статьи, представленные на 6-ом весеннем/летнем коллоквиуме молодых исследователей в области программной инженерии (SYRCoSE 2012), проводимом в Перми 30 и 31 мая 2012 г. Отбор статей производился на основе рецензирования материалов программным комитетом. На коллоквиум допускались как полные статьи, так и краткие сообщения, описывающие текущие исследования.

Программа коллоквиума охватывает следующие темы: технологии разработки программ; языки программирования, тестирование и верификация компьютерных систем; анализ программ; информационные системы, обработка изображений и видео и др.

ISBN 978-5-91474-019-8

Contents

Computer Networks and Telecommunication Protocols

Computer Science

Dynamic Analysis of Programs

Information Systems and Data Mining

Image and Video Processing

Application-Specific Methods and Tools

Foreword

Dear participants, we are glad to meet you at the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE). This year's event is held in Perm, a major administrative, industrial, scientific, and cultural center. The colloquium is hosted by the Perm branch of National Research University – Higher School of Economics (NRU HSE), one of the most prestigious universities in Russia. SYRCoSE 2012 is organized by Institute for System Programming of the Russian Academy of Sciences (ISPRAS) and Saint-Petersburg State University (SPbSU) jointly with NRU HSE.

Over the past years, Software Engineering (SE) has becoming a more mature scientific and technical discipline. However, being rapidly developed (as many others computer-related fields of knowledge), it cannot clearly identify its methods, object(s) of research and relation with other disciplines. Thus, in particular, it is not easy to determine a particular domain of some SE research. Such "fuzziness" has a certain influence on conferences and workshops on SE and information technologies in whole. SYRCoSE is not an exception. These proceedings contain not only papers on software development and analysis (SE in a restricted sense), but application-specific works as well (e.g., image processing and data mining).

In this year, Program Committee (consisting of 35 members from 22 organizations) has selected 40 papers. Each submitted paper has been reviewed independently by two or three referees. Participants of SYRCoSE 2012 represent well-known universities and research institutes such as A.P. Ershov Institute of Informatics Systems of RAS, Donetsk National Technical University (Ukraine), ISPRAS, Moscow Institute of Physics and Technology, Moscow State University, Novosibirsk State University, NRU HSE, NRU Saint-Petersburg State Polytechnical University, NRNU "MEPhI", Perm State National Research University, Russian-Armenian (Slavonic) University (Armenia), SPbSU, Saint-Petersburg NRU of Information Technologies, Mechanics and Optics, Taganrog Institute of Technology of Southern Federal University, Tomsk State University, Ural Federal University, Ural State University, Vladimir State University and Yaroslavl Demidov State University (3 countries, 11 cities and 19 organizations).

We would like to thank all the participants of SYRCoSE 2012 and their advisors for interesting papers. We are also very grateful to the PC members and the external reviewers for their hard work on reviewing the papers and selecting the program. Our thanks go to the invited speakers, Prof. Peter Gorm Larsen (Aarhus University, Denmark) and Prof. Nadezhda Yarushkina (Ulyanovsk State Technical University). We would also like to thank our sponsors, Russian Foundation for Basic Research (grant 12-07-06018-r), Microsoft Research, ICS and Prognoz. Finally, our special thanks to Galina Volodina (Director of NRU HSE – Perm), Valery Arkhipov (Deputy Director of NRU HSE – Perm), Lyudmila Lyadova and Vasilisa Korchagina for their invaluable help in organizing the colloquium in Perm.

Sincerely yours

Alexander Kamkin, Alexander Petrenko, Andrey Terekhov
May 2012

Committees

Program Committee Chairs

Alexander Petrenko – Russia
Institute for System Programming of RAS

Andrey Terekhov – Russia
Saint-Petersburg State University

Program Committee

Habib Abdulrab – France
National Institute of Applied Sciences, INSA-Rouen

Sergey Avdoshin – Russia
NRU Higher School of Economics

Eduard Babkin – Russia
NRU Higher School of Economics

Svetlana Chuprina – Russia
Perm State National Research University

Victor Gergel – Russia
Lobachevsky State University of Nizhny Novgorod

Efim Grinkrug – Russia
NRU Higher School of Economics

Maxim Gromov – Russia
Tomsk State University

Vladimir Hahanov – Ukraine
Kharkov National University of Radioelectronics

Shihong Huang– USA
Florida Atlantic University

Alexander Kamkin
Institute for System Programming of RAS

Vsevolod Kotlyarov – Russia
Saint-Petersburg State Polytechnic University

Oleg Kozyrev – Russia
NRU Higher School of Economics

Daniel Kurushin – Russia
State National Research Polytechnic University of Perm

Alexander Letichevsky – Ukraine
Glushkov Institute of Cybernetics, NAS

Irina Lomazova – Russia
NRU Higher School of Economics

Yury Lukach – Russia
Ural State University

Lyudmila Lyadova – Russia
NRU Higher School of Economics

Tiziana Margaria – Germany
University of Potsdam

Igor Mashechkin – Russia
Moscow State University

Valery Nepomniaschy – Russia
Ershov Institute of Informatics Systems, RAS

Elena Pavlova – Russia
Microsoft Research

Ivan Piletski – Belorussia
Belarusian State University of Informatics and Radioelectronics

Vladimir Popov – Russia
Ural State University

Yury Rogozov – Russia
Taganrog Institute of Technology, Southern Federal University

Nikolay Shilov – Russia
Ershov Institute of Informatics Systems, RAS

Ruslan Smelyansky – Russia
Moscow State University

Valeriy Sokolov – Russia
Yaroslavl Demidov State University

Vladimir Voevodin – Russia
Research Computing Center of Moscow State University

Dmitry Volkanov – Russia
Moscow State University

Mikhail Volkov – Russia
Ural State University

Rostislav Yavorsky – Russia
Witology

Nina Yevtushenko – Russia
Tomsk State University

Vladimir Zakharov – Russia
Moscow State University

Organizing Committee Chairs

Alexander S. Kamkin – Russia
Institute for System Programming of RAS

Lyudmila N. Lyadova – Russia
NRU Higher School of Economics

Referees

Sergey Avdoshin
Eduard Babkin
Igor Bourdonov
Mikhail Chupilko
Svetlana Chuprina
Denis Efremov
Rustam Galimulin
Victor Gergel
Efim Grinkrug
Maxim Gromov
Vladimir Hahanov
Shihong Huang
Alexander Kamkin
Olga Kondratyeva
Vsevolod Kotlyarov
Oleg Kozyrev
Daniel Kurushin
Natalya Kushik
Anna-Lena Lamprecht
Alexander Letichevsky
Irina Lomazova
Ludmila Lyadova
Tiziana Margaria
Valery Nepomniaschy
Elena Pavlova
Alexander Petrenko
Andrew Petukhov
Ivan Piletski
Vladimir Popov
Svetlana Prokopenko
Yury Rogozov
Natalia Shabaldina
Petr Shestov
Nikolay Shilov
Sergey Smolov
Valeriy Sokolov
Andrey Tatarnikov
Andrey Terekhov
Dmitry Volkanov
Mikhail Volkov
Rostislav Yavorskiy
Nina Yevtushenko
Vladimir Zakharov
Sergey Zelenov

Getting Software Engineering out of Isolation

(Invited Paper)

Peter Gorm Larsen and Joey W. Coleman
Aarhus University
Department of Engineering
Finlandsgade 22, DK-8200 Aarhus N, Denmark
e-mail: {pgl, jwc}@iha.dk

John Fitzgerald
Newcastle University
School of Computing Science
Newcastle upon Tyne, NE1 7RU, UK
e-mail: John.Fitzgerald@ncl.ac.uk

***Abstract*—We argue that the main challenges to be overcome in developing future generations of IT-enabled products and services lie not so much in the software engineering discipline itself as in the collaborative relationships that software engineers have with other disciplines. We briefly review the need for more emphasis on multi-disciplinary approaches and consider three classes of demanding system: embedded products, systems-of-systems and cyber-physical systems. In each of these areas, we argue that there is a need for engineering with formal semantic bases that enable joint modelling, analysis and simulation of groups of heterogeneous models.**

***Index Terms*—Formal methods, multi-disciplinarity, modelling, simulation, systems-of-systems, cyber-physical systems, embedded systems**

I. Introduction

The discipline of "software engineering", the origins of which are traceable to the NATO conferences of the late 1960s, emerged from a perceived need for a well-founded discipline underpinning a technical response to a software crisis [1], [2]. Unfortunately, the success rate of software development projects is still far from ideal [3], [4]. We would speculate that the rise of software engineering as a separate discipline has brought a real risk that software developers are distanced from the other engineering disciplines involved in product development. It almost seems a truism to say that software engineers need to understand well the system in which their product will operate, but the separation of engineering teams and phases in development still militates against this. The growth in embedded systems, and in networked integrations of software-rich systems, makes it imperative for software engineers to find ways of developing systems collaboratively with disciplines that may use quite radically different modelling and analysis techniques. This implies global system models that encompass cyber and physical elements, enabling the analysis and trade-off of decisions across the boundaries between these domains.

The mono-disciplinary nature of most engineering educations contributes to the challenge here. Graduates trained as experts in one single discipline may have difficulties in understanding stakeholders from different disciplines and the challenges that they face. As a result, solutions that are globally optimal from a system-wide perspective are not reached because modelling and analysis work tends to focus on mono-disciplinary optimisation. There is a need in our education system to ensure that students get at least a feel for the significant design parameters from other disciplines.

One example of such an opportunity for collaborative engineering is between control and software engineering. Modelling embedded systems that are intended to affect the physical world is a significant challenge [5]. Models of the physical world often involve differential equations that describe how the world changes over time; these models are based on continuous mathematical domains. Models of digital controllers –often software-based– usually do not reference time at all and instead treat the model of the controller in terms of discrete events that trigger specific reactions from the controller; these models are based on discrete mathematical domains [6]. In order to be able to appropriately balance concerns from control engineering with software engineering and taking potential faults into account a common model that can be analysed is needed [7]. The challenge can be further extended when different systems can provide added benefits for its users by interaction with other systems. There is a new emerging discipline for the proper engineering of such System of Systems (SoSs). Here the challenges cannot be solved using software engineering or system engineering principles alone. The scaling here gives new research challenges that clearly can get inspiration from existing engineering disciplines but a multidisciplinary approach is necessary again.

In this paper we look at the need for multidisciplinarity in engineering education in Section II. In Section III, we focus on how software engineering aspects might be combined with consideration of the physical world surrounding software. Section IV discuss scaling software development up to the level of "Systems-of-Systems". Section V takes this further on to cyber-physical systems that have aspects of both embedded systems and systems-of-systems. Section VII provides concluding remarks and identifies future research directions aiming at getting software engineering out of isolation.

II. Multidisciplinary Engineering Education

Most university studies are highly specialized within a single discipline –to the point of being strictly mono-disciplinary at some universities– in order to impart a sufficient depth of skill to the students. This approach is true of most European universities and many North American institutions as well.

As a contrast, there are universities, mostly in North America though some are in Europe, that follow the liberal arts tradition. These institutions require that students take courses from outside of their chosen discipline. This applies mostly to programs offered at the Bachelour's level, as Master's level degrees remain specialized even at liberal arts institutions.

At all levels of education, however, it is important that the students have some significant exposure to challenges and terminology from other disciplines [8]. Without this exposure there is a substantial risk that students starting their careers will face challenges in understanding and collaborating with peers and stakeholders from different disciplinary backgrounds. The question is how to best prevent this situation from happening, and how to provide the appropriate tools for students to overcome this situation when it does occur. Practical experience suggests the direction of a broader answer to this question.

A summer school with the aim of providing students with a multi-disciplinary angle to their studies has run since 2008 in Denmark, in collaboration with Bang & Olufsen [9]. The summer school is entitled "Conceptual Design and Development of Innovative products". Students with different disciplinary and cultural backgrounds are combined in mixed groups. Each group then receives assignments that require them to use their combined skills to solve it appropriately. This requirement for collaboration is rooted in problem-based learning principles [10]. The results of delivering this summer school series has been so successful that the idea is being exported and expanded with a similar summer school in China in 2012.

A similar multidisciplinary summer school for PhD students was delivered for the first time in Portugal in 2012, titled "Innovation and Creativity for Complex Engineering Systems". Here the focus was more on producing multidisciplinary research plans. This summer school is also project-based but more teacher-led lecturing was included to give the students a better understanding of specific topics, such as how to write multidisciplinary research plans.

The challenge is to get such stand-alone events incorporated into the standard curriculum used in (engineering) educations such that all students get exposed to this kind of experience during their studies. We believe that a first start of this is to ensure that all (engineering) students get a joint course on system engineering [11]. However, we believe that collaboration across disciplines and respect for the challenges in other disciplines will only fully be achieved if the students try to solve multi-disciplinary assignments in multi-disciplinary groups.

In the spirit of the liberal arts traditions seen in Bachelor's education, we would ultimately like a similar "technical arts" tradition for specialist Master's degrees. This is most important for those programs whose topic inevitably leads to cross-disciplinary collaboration. This proposal is not to introduce a "general science year" into the curricula, but rather to ensure that the graduates of specialist Master's programs have significant exposure to the terminology and challenges of other scientific disciplines.

The incorporation of a stand-alone multidisciplinary project course –tacking problems that require input and skills from many disciplines– into the regular curriculum is a first step towards a technical arts tradition. The main challenges for implementing these courses are mainly of practical administration since such courses typically need to be coordinated between different disciplinary studies and across different departments. However, if it was successfully implemented the students would get out of isolation of their own discipline and as a consequence be much better at solving more complicated challenges at the general system level when they would enter their professional careers.

III. Embedded Systems

An embedded system is a computer system designed for specific control functions within a larger system, often with real-time computing constraints [12]. It is embedded as part of a complete device often including hardware and mechanical parts and typically with less interface towards human users. By contrast, a general-purpose computer, such as a Personal Computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

The DESTECS project[1] has taken a first step in the direction of crossing between different disciplines necessary in the embedded control domain [13]. This project addresses collaborative, multidisciplinary design of embedded systems using methodology and tools that promote rapid construction and evaluation of well-founded system models.

One of the main impediments to the design of embedded real-time control solutions is the separation of engineering disciplines. While control engineering typically uses tools operating on Continuous-Time (CT) models, software engineering is founded on Discrete-Event (DE) models. In order to evaluate alternative designs and support early defect analysis or correction, it is essential that engineers collaborate across disciplines in short windows of opportunity [14], [15]. Model-based approaches provide a way of encouraging collaboration, but engineers need to jointly perform design evaluation and analysis using models expressed in different tools. These tools should reflect the relevant aspects of the design in a natural way, but also allow consistent, rapid analysis and comparison of models. Achieving this requires advances in CT modelling; formal DE modelling of controllers and architectures; fault modelling and fault tolerance; and open tools frameworks. These various advances are the aim of the DESTECS project.

An Example: Train Carriage Braking

It is simply impossible to develop a useful controller without close interaction between the disciplines of control engineering, software engineering, mechanical engineering and electrical engineering. Each of these will have their own established modelling techniques and formalisms. As an example, consider the development of software for controlling the speed of railway carriages by applying the brakes. Associated with

[1] "Design Support and Tooling for Embedded Control Software" http://www.destecs.org

each carriage, control software takes account of environmental conditions (e.g. current speed, temperature, friction etc.) and fixed design parameters (e.g. number and position of wheels, mass of the carriage etc.) in order to determine how best to apply the brakes on command from the driver or safety unit. It only makes sense to talk about the behaviour of this software in the context of the product of which it is a part – the railway carriage as a whole.

Imagine one scenario for the development of the braking system. Well-established control laws for this type of braking system are developed using tools based on continuous time models (perhaps using numerical solutions to differential equations as a simulation). These laws are passed to software developers who discover that the laws cannot be directly implemented on the processors available because certain calculations necessary for processing the data from the sensors cannot be completed quickly enough within the processor schedule. By this stage in the development process it may be too late to modify the carriage design, or use alternative and better sensors. The CPUs for the controller software will often have been fixed and even purchased some time previously, so they cannot be replaced with faster processors. The only remaining option may be to modify the schedule running on the processor, so that some other functionality affecting less critical functions, like the smoothness of the ride, for example, have to be rescheduled, compromising performance and the quality of the product. In practice, there can often be a slow iterative process in which control laws are re-engineered and re-implemented several times before a compromise is reached, reducing time to market.

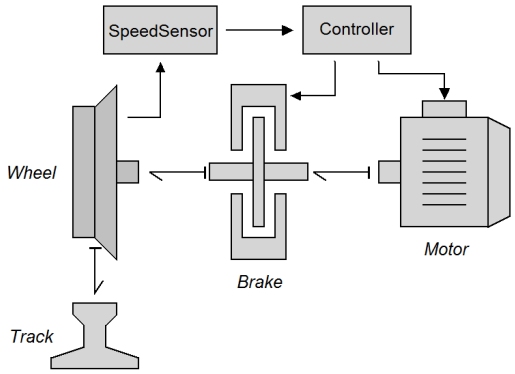


Fig. 1. A 20-sim model of a train carriage braking built from bond graphs and iconic diagrams.

If the developers could model the control laws collaboratively and *at the same time as* the software, some of these difficulties might be reduced. For example, an alternative choice of sensor might be made, replacing the original design with one that does pre-processing of the data, and this could be checked out and evaluated at the modelling stage. In DESTECS, we build such collaborative models (*co-models*) which represent the semantic integration of the software design, based on DE computation models in VDM, with the CT models of the plant and control laws in a bond graph formalism supported by the 20-sim environment (see Figure 1). For our example, such a co-model would contain:

- A CT model encompassing the wheels of the carriage, their friction and force linkages to the track, and the braking mechanism itself.
- A DE model of the control software, including the main control loop. This may also include the supervisory control software which manages system functionality a level above the loop control, including the switching of modes, error detection and recovery. In our experience, supervisory control accounts for 80% of the software content of an embedded product, and can be a greater source of potential defects than the loop controller itself. The DE model can readily be expressed in VDM with its real-time extensions. In our example, supervisory control might be responsible for switching alternative control laws into force if the temperature of the brakes exceeds a certain value, or invoke emergency braking modes if signalled to do so from the train driver's cab.
- An interface (in DESTECS this is termed the *contract*) between the two models defining the shared design parameters that both sides need to know, the shared variables that are monitored or controlled and any events of interest. Events are logical predicates that may cause the simulation of the CT-side model of the plant to be interrupted in order so that a response can be generated by the controller (for example if a sensed value crosses a threshold).

The question of where to place the models of sensors and actuators is interesting. In many cases it is appropriate to describe them "CT-side", but digital control might sometimes suggest placing them DE-side. The co-model as a whole presents an interface to a co-simulation engine that allows the CT and DE models to be executed together. The co-simulation engine implements a reconciled operational semantics of the two models, managing the synchronisation of time and state between them. The co-simulation can proceed under the control of a script that implements a particular scenario in terms of the actions of the environment (for example in raising a braking signal), and the invocation of fault models that may be built in to the DE or CT models. The exploration of the space of design alternatives is enabled by such co-simulation. Multiple scenario-based tests can be used to assess the performance of either alternative plant or controller designs. In our example, this could include an assessment of alternative numbers and configurations of sensors (modelled CT side) with a appropriate changes of control loop (DE-side). Being able to perform design space exploration at an early stage is the essence of successful system design, and the example emphasises the extent to which this is a multi-disciplinary activity enabled by software engineers work in collaboration with others, and certainly not alone. The challenges that remain in such cross-disciplinary assignments are much more complex and important to tackle than those that remains inside software engineering itself.

IV. Systems-of-Systems

Modern network technologies are enabling the integration of pre-existing computing systems into "Systems-of-Systems" (SoS) that together deliver a service that the constituent systems could not offer alone [16]. Examples include emergency response systems formed from the coalition of information systems of the response services such the ambulance, hospital and fire services. The public who expect responsive emergency services may demand confidence that the SoS will deliver safe, rapid and secure transfer of patient data between these systems. Examples on another scale might include the audio-video ecosystem in a home in which digital content is streamed from multiple sources to multiple users via a range of systems provided by different manufacturers. The customer experiencing the SoS expects to have a consistent experience (such as a common playlist) as they move through the ecosystem from device to device. The manufacturers of the devices also need to demonstrate that they will respect the digital rights agreements in force for the content as it is played through their devices, even though they may be delivered through another system in the SoS.

While embedded systems are often characterised by closed loop control, SoS are more general, are distributed, and typically have more human interaction. The distinguishing characteristics of SoSs are:

- **Operational independence:** A SoS is formed by heterogeneous constituent systems, many of which may not have been originally designed for participation in the SoS. They may be described using a wide range of methods and require modification, for example, through wrapping or linking interfaces, in order to achieve integration
- **Managerial independence:** The constituent systems may be managed independently and so can change functionality or character during the life of the SoS in ways that are not foreseen when they are originally composed.
- **Distribution:** Constituent systems may be distributed and decoupled, and yet a communications infrastructure should support the protocols necessary to facilitate coordination between them.
- **Evolutionary development:** The independence of constituent systems means that the SoS changes over time to respond to changing goals or component characteristics.
- **Emergence:** SoSs exhibit behaviour that their components do not exhibit on their own.

In a rather conventional view of software engineering, software systems are constructed in a highly directed way by teams who are usually within the same organisation, and who (at least on paper) have a shared understanding of the goals of the system being developed. The components can be designed to use carefully defined interfaces that violate their independence by revealing data and services in order to manage their collaboration. The operational and managerial independence characteristics mean this is not the case. One category of SoS ("directed" SoS [17]) does allow for a master that has the power to get owners of other constituent systems to adapt to their wishes, but SoS that adhere to this structure are comparatively rare. In general, the characteristics listed above pose major challenges to conventional "closed" software engineering methods if we wish to develop SoS that are dependable. Independence means that developers can have only limited knowledge of, and confidence in the likely behaviour of constituent systems. Distribution (in some cases also mobility) can compound the difficulty of gaining confidence in concurrent behaviours. The need to manage evolutionary development means that some ability to cope with change must be built-in. Above all the reliance on emergence means that the verification of global SoS-level behaviour must follow from the composition of the behaviours of individual constituents. As our emergency response and audio-video examples show, in practical terms, SoS Engineering is challenged by the large number and range of stakeholders (the owners and operators of the constituent systems as well as the users who experience the SoS as a whole). Here again, software cannot be developed in isolation and thus software engineers need to be able to think in a SoS setting where they cannot control all parts.

The COMPASS[2] project that aims to develop systematic engineering principles that are applicable to SoS design, including the scaling-up of modelling and validation techniques from a formal methods to address the SoS challenges. COMPASS is defining the first formally based modelling language specially designed to target the SoS area [18]. The challenge of constituent system independence is addressed by recording contracts that express our limited knowledge about constituents, constraining, but not completely prescribing, their range of behaviours. Global SoS-level properties are verified as the composition of the properties guaranteed in the constituent system contracts.

Example: Interoperable Train Carriage Systems

An important objective of development work in the rail sector is to increase the interoperability of railway equipment such as carriages from different suppliers, so that it becomes possible to mix them in the same "set" or train, for example by coupling trains from different railway systems at national borders, to form larger trains.

A train made up from a heterogeneous collection of component carriages is a form of a SoS, and Figure 2 gives a partial representation of this. The constituent systems are the information and computing systems in each carriage. These constituents were not necessarily designed with the intention of being in a mixed set, and they will generally be running software that is managed and upgraded by suppliers who are independent of each other. They are networked in the train and geographically distributed, and the software in each manufacturer's carriage can change with time. In spite of all this, we expect them to deliver a consistent emergent experience to the passengers on board. The developers of train systems are unlikely to subjugate their own corporate

[2]This is an acronym for "Comprehensive Modelling for Advanced Systems of Systems".

goals to those of the SoS, so this is not a "directed" SoS. We would not risk simply plugging carriages together, so some level of explicit cooperation is needed, even at the level of ensuring data transfer, so this is not a "virtual" SoS either [16]. Rather, depending on the degree of explicit cooperation, it is an "acknowledged" or a "collaborative" SoS.

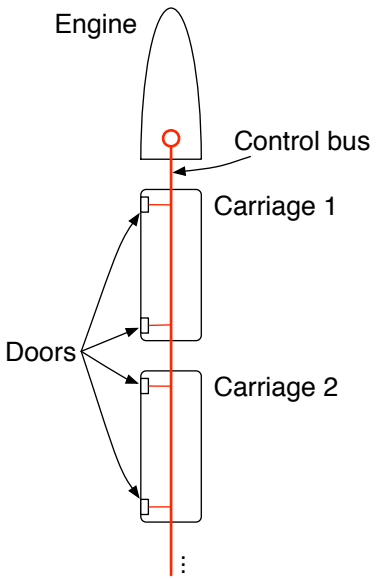


Fig. 2. Diagram of a system of train carriages.

A model of this SoS contains contractual descriptions for each of the services offered at the boundaries of each carriage and other constituent systems. Such contracts involve assumptions and guarantees. For example, the contract on a door locking service of a constituent system might record a guarantee to lock the doors within 3 seconds of receiving a specified signal; the assumption might be that the signal comes with a valid carriage identifier. Contracts also record the constraints on interaction behaviours, recording for example that a "Lock the doors" command is acknowledged once the doors have been locked, or a special response denoting failure if the locks did not work. Furthermore, the door locking services of all carriages must communicate with the control bus on each carriage, and each carriage's control bus must cooperate with the busses on connected carriages and with the control system in the engine.

In COMPASS, we work on formalisms for contracts that support the description of both functional and interaction behaviours. Verifying a global property, such as the fact that all of the train's doors will be locked, or failure acknowledged within a specified time, should follow from the contracts offered by the constituents. For such a SoS we require not merely that data should be syntactically compatible between units, but that there should be a semantic mapping, so that "Lock the doors!" is interpreted in similar ways in all carriages of the train. In SoS engineering, as in embedded systems, software engineers have to deal with heterogeneity of systems and models, and cannot confine their analysis to mono-disciplinary approaches. This again provides a need to think outside the isolated software engineering discipline.

V. Cyber-Physical Systems

Cyber-Physical Systems (CPS) are integrations of multiple computing and physical processes, with the potential to design and adapt both computing and physical elements to improve efficiency and resilience of the system as a whole [19], [20]. This encompasses the conventional control systems which typically are represented in a static setup. However CPS can also be seen in increasingly dynamic settings. Examples of mobile cyber-physical systems include applications to track and analyse Carbon Dioxide emissions [21], detect traffic accidents and provide situational awareness services to first responders [22], measure traffic [23], and monitor cardiac patients [24].

The engineering challenges associated with developing dependable CPS combine those of embedded systems and systems-of-systems. The ultimate goal here is to be able to analyse trade-offs of design alternatives that cross the boundaries between cyber and physical elements, as with embedded systems, but also between multiple cyber and multiple physical elements. As with SoS, the integrator has only limited knowledge and confidence about the constituent computing and physical elements, so these can only be modelled in contractual terms. In a CPS setting, we need to consider a range of interfaces between physical and cyber elements, including force or other physical phenomena, and not merely data. Indeed, the field of rigorous engineering methods for dependable CPS is still in its infancy [25].

Example: Controlling the Train!

In our multi-carriage train example, the SoS formed by distributed control of the diverse carriage braking systems forms a CPS. Developers requiring to verify that braking commands will result in the train speed reducing within a specified distance or time need to consider the multiple cyber control elements in the constituent carriages, and their different capabilities, as well as the effects of braking on the train physics. In this latter aspect, the physical elements have an influence on one another through the braking force of the train. Aside from verifying safety-related properties, having a CPS multi-model based on a number of networked co-models, allows us to analyse properties such as energy consumption in different operating scenarios. The kinds of design trade-off that might be considered are variations in the braking command parameters sent to different carriages to take account of their physical differences as well as the differences in their control characteristics. Aside from the multiple co-modelling involved, there is a significant amount of research to be done in visualisation of scenario outcomes, and the associated support for design space exploration, in such a complex system.

VI. Related Work

The ideas presented here build for example on advances in embedded systems design and fault modelling. In the embedded sector, BODERC [26] developed a method to predict

performance of real-time control systems, albeit with little tool support for trade-off studies or co-simulation. Modelica [27] is an object-oriented, equation-based multi-domain language for simulating controlled physical systems, and provides source libraries of physical components similar to the approach taken in the DESTECS project. Approaches to co-simulation of discrete-event and continuous-time models have been defined by Nicolescu et al. [28]. Ptolemy II [29] offers both discrete-event and continuous-time simulation within a single tool, though lacking the object-orientation offered by VDM and the component libraries offered by 20-sim. Work on time synchronisation between DE and CT models is described in *hybrid systems* literature, for instance, Cassandras et al. [30]. The DESTECS approach is distinctive in including a rich but abstract DE-side modelling language, and in managing co-simulation of heterogeneous models in their "native" tools.

Collaborative modelling is essential in both SoS and CPS engineering. Several approaches to collaborative modelling are described by Renger et al. [31]: problem structuring methods focus on the decision-making process including simulation for scenario exploration; group model building takes extends the conceptual model to simulation models to explore different options; enterprise analysis focuses on models that are built collaboratively. Our work is focussed on collaborative construction of models that are then used to explore design options.

VII. Concluding Remarks

Software engineering is a maturing discipline with sound scientific foundations, a range of methodologies, increasingly robust tools, and a wealth of experience. However, in this paper, we have argued that the complex products, solutions and services developed in the future, and the levels of dependability that they demand, require a more multidisciplinary and collaborative approach. This has consequences for software engineering technology, for formal methods themselves, and for training and education, all of which need to cross an increasingly broad range of disciplines and modelling types.

Our experience in the DESTECS and COMPASS projects has been a first small step in this direction. There remain opportunities for significant advances in modelling technology, including the areas of semantics, tool support, design space exploration, methodology and model management. The need to package relevant research results as industry-ready methods and (open) tools is paramount. We hope that the ideas presented in this paper will cause more researchers to carry out their research in software engineering in a larger context in order to have a more significant impact on the actual development of software-based systems in industry. Hopefully there will be fruitful collaborations for this kind of multidisciplinary research.

Acknowledgements

Both the DESTECS and the COMPASS projects have been supported by the European Commission under the 7th Framework programme. We would like to thank our collaborators from the DESTECS and COMPASS projects for their work to make the joint visions become reality. Finally we would like to thank Nick Battle, Carl Gamble and Claus Ballegaard Nielsen for providing valuable input on drafts of this paper.

References

[1] P. Naur and E. B. Randell, "Software Engineering: Report on a Conference sponsored by the NATO Science Committee," garmisch, Germany, 7th to 11th October 1968, Brussels, Scientific Affairs Division, NATO, January 1969.

[2] W. W. Gibbs, "Software's Chronic Crisis," *Scientific American*, pp. 72–81, September 1994.

[3] The-Standish-Group, "The Chaos Report," http://www.projectsmart.co.uk/docs/chaos-report.pdf, 1995.

[4] J. Johnson, *My Life Is Failure: 100 Things You Should Know to Be a Better Project Leader*. Standish Group International, 2004.

[5] T. Henzinger and J. Sifakis, "The Discipline of Embedded Systems Design," *IEEE Computer*, vol. 40, no. 10, pp. 32–40, October 2007.

[6] A. Tiwari, N. Shankar, and J. Rushby, "Invisible Formal Methods for Embedded Control Systems," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 29–39, January 2003.

[7] M. Verhoef, "Modeling and Validating Distributed Embedded Real-Time Control Systems," Ph.D. dissertation, Radboud University Nijmegen, 2009.

[8] S. Jahanian and J. M. Matthews, "Multidisciplinary Project: A Tool for Learning the Subject," *Journal of Engineering Education*, April 1999.

[9] P. G. Larsen, J. M. Fernandes, J. Habel, H. Lehrskov, R. J. Vos, O. Wallington, and J. Zidek, "A Multidisciplinary Engineering Summer School in an Industrial Setting," *European Journal of Engineering Education*, August 2009.

[10] D. L. Maskell and P. J. Grabau, "A Multidisciplinary Cooperative Problem-Based Learning Approach to Embedded Systems Design," *IEEE Transactions on Education*, vol. 41, no. 2, pp. 101–103, May 1998.

[11] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *System Engineering – Coping with Complexity*. Pearson Education, 1998, vol. ISBN 0-13-095085-8.

[12] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design, Modelling Synthesis and Verification*. Springer, 2009.

[13] J. F. Broenink, P. G. Larsen, M. Verhoef, C. Kleijn, D. Jovanovic, K. Pierce, and W. F., "Design Support and Tooling for Dependable Embedded Control Software," in *Proceedings of Serene 2010 International Workshop on Software Engineering for Resilient Systems*. ACM, April 2010, pp. 77–82.

[14] J. Fitzgerald, P. G. Larsen, K. Pierce, M. Verhoef, and S. Wolff, "Collaborative Modelling and Co-simulation in the Development of Dependable Embedded Systems," in *IFM 2010, Integrated Formal Methods*, ser. Lecture Notes in Computer Science, D. Méry and S. Merz, Eds., vol. 6396. Springer-Verlag, October 2010, pp. 12–26.

[15] J. Fitzgerald, P. G. Larsen, K. Pierce, and M. Verhoef, "A Formal Approach to Collaborative Modelling and Co-simulation for Embedded Systems," *To appear in Mathematical Structures in Computer Science*, 2012.

[16] M. W. Maier, "Architecting Principles for Systems-of-Systems," *Systems Engineering*, vol. 1, no. 4, pp. 267–284, 1998.

[17] J. Dahmann and K. Baldwin, "Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering," in *IEEE Systems Conference*. IEEE, April 2008.

[18] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, "Features of CML: a Formal Modelling Language for Systems of Systems," in *The 7th International Conference on System of System Engineering*, July 2012.

[19] J. White, S. Clarke, C. Groba, B. Dougherty, C. Thompson, and D. C. Schmidt, "R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services," *J. Internet Services and Applications*, vol. 1, no. 1, pp. 45–56, 2010.

[20] E. Lee and S. Seshia, *Introduction to Embedded Systems, A Cyber-Physical Systems Approach*. University of Berkley: http://LeeSeshia.org, 2011, iSBN 978-0-557-70857-4.

[21] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harrison, and J. A. Landay, "UbiGreen: Investigating a Mobile Tool for Tracking and Supporting Green Transportation Habits," in *Proceedings of the 27th international conference on Human factors in computing systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 1043–1052.

[22] C. Thompson, J. White, B. Dougherty, and D. C. Schmidt, "Optimizing Mobile Application Performance with Model-Driven Engineering," in *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, ser. SEUS '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 36–46.

[23] G. Rose, "Mobile Phones as Traffic Probes: Practices, Prospects and Issues," *Transport Reviews*, vol. 26, no. 3, pp. 275–291, 2006.

[24] P. Leijdekkers and V. Gay, "Personal Heart Monitoring and Rehabilitation System using Smart Phones," in *Proceedings of the International Conference on Mobile Business*. Washington, DC, USA: IEEE Computer Society, 2006.

[25] H. Giese, B. Rumpe, B. Schätz, and J. Sztipanovits, Eds., *Science and Engineering of Cyber-Physical Systems (Dagstuhl Seminar 11441)*, ser. Dagstuhl Reports. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, vol. 1(11).

[26] M. Heemels and G. Muller, *Boderc: Model-Based Design of High-tech Systems*, 2nd ed. Den Dolech 2, Eindhoven, The Netherlands: Embedded Systems Institute, March 2007.

[27] P. Fritzson and V. Engelson, "Modelica - A Unified Object-Oriented Language for System Modelling and Simulation," in *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*. Springer-Verlag, 1998, pp. 67–90. [Online]. Available: http://www.modelica.org/documents/ModelicaSpec32.pdf

[28] G. Nicolescu, H. Boucheneb, L. Gheorghe, and F. Bouchhima, "Methodology for Efficient Design of Continuous/Discrete-Events Co-Simulation Tools," in *High Level Simulation Languages and Applications*, J. Anderson and R. Huntsinger, Eds. San Diego, CA: SCS, 2007, pp. 172–179.

[29] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming Heterogeneity – the Ptolemy Approach," *Proc. of the IEEE*, vol. 91, no. 1, pp. 127–144, January 2003.

[30] C. G. Cassandras, *Analysis and Design of Hybrid Systems: a Proceedings Volume from the 2nd IFAC Conference*. Elsevier, Jun. 2006.

[31] M. Renger, G. L. Kolfschoten, and G.-J. Vreede, "Challenges in Collaborative Modeling: A Literature Review," in *Advances in Enterprise Engineering I*, ser. Lecture Notes in Business Information Processing, J. L. G. Dietz, A. Albani, J. Barjis, W. Aalst, J. Mylopoulos, M. Rosemann, M. J. Shaw, and C. Szyperski, Eds. Springer Berlin Heidelberg, 2008, vol. 10, pp. 61–77.

Time Series Analysis by Soft Computing Methods

N. Yarushkina
Information Systems Department
Ulyanovsk State Technical University, UlSTU
Ulyanovsk, Russia
jng@ulstu.ru

Abstract—**Qualitative evaluation and comparison of changes of indications of objects having different nature is used by designers, managers, people making decisions (PMD) and experts to make the decisions more reasonable. For suport of such activity on the analysis of changes of data connected with certain dates and time intervals, models of fuzzy time series are applied. In this article a model of fuzzy tendency the carrier of which is a fuzzy time series and its variety — elementary tendency model — is offered. The offered models are aplied for solution of the problem of summarization of fuzzy time series in terms of tendencies.**

Keywords Fuzzy Time Series, Fuzzy Tendency, Elementary Tendency, Summarization.

I. INTRODUCTION

In connection with increasing volume and speed of storage of data connected with certain dates and time intervals in data bases, the new technology of data analysis Data Mining for Time Series Data Base (TSDM) is actively developed. Forming these time sequences on the basis of domains permits to consider them in the form of time series (TS). A number of distinctions of such time sequences from traditional (classic) TS, considered in statistic theory of analysis and forecast of TS, can be noted: these time sequences are short TS; such time sequences can be represented not only by numerical but also by linguistic values; it is difficult or impossible to determine suppositions about random nature of values for such time sequences. Distinctive properties of Time Series Data Base generate changes in the aggregate of aims and tasks which form the direction Data Mining for Time Series Data Base[1]: segmentation, clusterisation, classification, indexing, summarization, disclosing anomalies, frequency analysis, forecasting, extraction of associative rules. At present the methodology of solving the indicated problems is formed on the basis of methods and models of intelligent analysis, representation and processing of data [2, 3, 4, 5]. The basis of the offered methods of intelligent data analysis is the concept of the fuzzy time series (FTS) constructed on the base of levels of TS [5, 6] or their first differences [3]. In this article an original model of FTS on the basis of fuzzy tendencies is offered. The offered model describes the behaviour of FTS in the form of the sequence of fuzzy tendencies represented by information granules. Application of information granules for problems of Data Mining for Time Series Data Base is presented in the work [7]. An indisputable advantage of data granulation is the possibility of representation of models of TS at different levels of abstraction in the linguistic form, which permits to widen the accessibility of using models of TS by application users and to improve their interpretability.

In the second part of the article models of a fuzzy time series and a fuzzy tendency are considered. In the third part a special scale for generating an elementary fuzzy tendency is given. Procedures of granulation of FTS in the basis of FT are presented in the fourth part, there is also described the solution of the problem of summarization of TS in terms of FT and presented the experiment on the solution of the problem of summarization of a synthetic TS as the problem of determination of a fuzzy tendency.

II. FUZZY TENDENCY MODEL

A. Concept of linguistic evaluation

One of the problems of the analysis of TS is the analysis of FTS behaviour, that is change of values of TS levels. The solution of the problem of the analysis of TS behaviour expressed in linguistic form can serve as the linguistic evaluation of the behaviour. Linguistic evaluations (LE) are the means of qualitative evaluation and comparison of characteristics and indications of objects having different nature used by designers, managers, people making decisions (PMD), experts. An important property of linguistic evaluations is wide application in practice in making decisions for expressing knowledge about the degree of correspondence of the object being observed or its characteristics to some objective or subjective criterion. The stated property determines the class of absolute LE, which reflects the static aspect of evaluation. The following expressions can be examples of such evaluations: "Satisfactory", "Good", "Bad", "Big", "Small", "Medium", etc. The semantics of absolute linguistic evaluations depends on the context of the environment in which they are used and modeled by fuzzy sets.

Another important property of linguistic evaluations is conditioned on the possibility of ranking them, it permits to present the aggregate of LE in the form of some system with relations. Binary relations formed on the set of absolute LE generate comparative linguistic evaluations by different criteria such as "More", "Less", "Approximately Equal", "Earlier",

"Later", "Rather", "Better", etc. Comparative evaluations made on the basis of absolute LE can represent changes in different universes: in the universe of objects, in the time universe, in the universe of problems and they express dynamic aspect of evaluation. The semantics of comparative evaluations is also context-dependent and can be modeled on the basis of fuzzy sets.

It is noted in the article [4] that linguistic evaluation has indications expressing the degree of intensity of this evaluation. These indications can be represented in the linguistic form, usually used by people: "Very", "Insignificantly", "Approximately", etc.

Context-dependent linguistic evaluations considered above are given in expert way as a rule, and they are called expert evaluations. In case of impossibility of receiving expert evaluations of indications of objects, abstract linguistic evaluations are used, let us consider such evaluations among the class of context-free linguistic evaluations.

Let the aggregate of all linguistic evaluations forms the finite set $\tilde{X} = \{\tilde{x}_j\}$, where $j \in [1,nj]$. Let us call the linguistic evaluation of the indication x from the universe B a fuzzy label, if the fuzzy set is determined for it such that $\tilde{x}_j = \{\langle w_m, \mu_{\tilde{x}_j}(w_m)\rangle \mid w_m \in w, w \subseteq B\}$, where w_m – is the carrier of the fuzzy set, $\mu_{\tilde{x}_j}(w_m)$ – is the membership function.

B. Model and kinds of a fuzzy tendencys

Let us introduce definitions.

Let some time series (TS) $Xt=\{t_i,x_i\}$ is given, $i \in [1,n]$, n – is the quantity of members of the series, $x_i \in B$, $t_i \in Bt$.

Let us call the ordered sequence of observations over some phenomenon the states of which change in time if the value of the state at the instant t_i is expressed with the help of the fuzzy label $\tilde{x}_i \in \tilde{X}$, $i \in [1,n]$, n – is the quantity of members of series, a *fuzzy time series (FTS)*. That is we represent a fuzzy time series in the form $\tilde{X}_t = \{t_i, \tilde{x}_i\}$, where $\tilde{x}_i$ – i-th fuzzy set (fuzzy label), t_i – i-th value of the instant of time, $t_1 \le t_i \le t_n$, n – is the quantity of members of FTS. Any TS can be represented in the form of the sequence of fuzzy labels $\tilde{x}_j = \{\langle w_m, \mu_{\tilde{x}_j}(w_m)\rangle \mid w_m \in w, w \subseteq B\}$ on the basis of linguistic (context-dependent or context-free) evaluation of levels of TS $x_i \in B$.

Let us call the fuzzy label $\tau_k \in \tilde{X}$ expressing the character of change (systematic motion) of the sequence of fuzzy values of FTS $\tilde{X}_t$ in the given interval of time the *fuzzy tendency (FT)* of a fuzzy time series. A fuzzy tendency determines the nature of FTS not in analytic, but in the linguistic form.

Each fuzzy tendency τ_k of the fuzzy time series $\tilde{X}$ can be represented by the fuzzy set $\tau_k = \{\langle \tau i, \mu_{\tilde{X}_t}(\tau i)\rangle, i \in [1,n]\}$ with the function of membership in the fuzzy time series $\mu_{\tilde{X}_t}(\tau i)$, where τi is the model of the following form:

$\tau i = \langle \nu_i, \alpha_i, \Delta t_i \rangle$, where

ν_i – is the type of the tendency. Let us compare fuzzy labels τ_k and base types of tendencies "Increase", "Decrease", "Stability". On the basis of base types derivative types of tendencies, such as "Fluctuations", "Chaos", "Load", "Idle time", etc., can be formed.

α_i– intensity of the tendency.

Δt_i – duration of the tendency.

In case if $\Delta t_i = 1$, let us consider the fuzzy tendency among the class of elementary fuzzy tendencies, if $\Delta t_i = n-1$, we consider the fuzzy tendency among the class of general fuzzy tendency of FTS, if $1 < \Delta t_i < n-1$, let us consider the fuzzy tendency among the class of local (derivative) fuzzy tendencies of FTS.

In the aspect of content an elementary fuzzy tendency models changes between two neighbouring values of the fuzzy time series $\tilde{x}_i, \tilde{x}_{i+1}$, and it can be compared with the instant of time t_i of FTS. A local fuzzy tendency is determined between two chosen values of the fuzzy time series $\tilde{x}_i$ and $\tilde{x}_j$, when $i < j$ can be compared with the instant of time t_i. Any local FT can be expressed by the sequence of elementary ET. The general fuzzy tendency characterizes the behaviour of all FTS and it is representable in the form of the sequence of local and, therefore, also elementary FT. Thus, the time series of elementary FT of the form $\{t_i, \tau_k(t_i, \tilde{x}_i)\}$, $i \in [1,n-1]$ can be made for any FTS.

The analysis of fuzzy labels used when evaluating levels and behaviour of a time series permits to make the following conclusions:

1. Models of elementary, local, general FT of a fuzzy time series have common structure.
2. Local and general fuzzy tendencies of FTS can be expressed through the time series of elementary FT.
3. A time series of elementary FT is an invariant method of linguistic representation of behaviour of any FTS.
4. Representation of time series in the form of fuzzy time series and time series of elementary fuzzy tendencies permits to take into account additional knowledge in the form of semantics of application area during their analysis owing to use of context-dependent fuzzy labels.

In the next part we will introduce a special scale for generation of the model of the elementary fuzzy tendency.

III. ACL-SCALE FOR GENERATION OF THE MODEL OF THE ELEMENTARY TENDENCY

In this part a special linguistic scale is offered as a tool of both absolute and comparative linguistic evaluation – *ACL-scale (Absolute & Comparative Linguistic)*. This scale will be

applied for construction of the model of the elementary fuzzy tendency.

The model of ACL-scale S_x for determination of absolute and comparative linguistic evaluations is representable in the form of linguistic variable with relations

$$S_x = <Name\_S_x, \tilde{X}, B, G, P, TTend, RTend>,$$

where $Name\_S_x$ – is the name of ACL-scale, $\tilde{X}$ – is the base term-set of absolute LE (linguistic name of gradations), for example, $\tilde{X}$ ={Bad, Satisfactory, Good, Excellent, ..}, $\tilde{x}_i \in \tilde{X}$, B – is the universal set on which the scale is determined, $x \in B$. G – are syntactic rules of deduction (generation) of chains of evaluation propositions (derivative of terms which do not enter into the base term-set), P - are semantic rules which determine membership functions for each term (they are usually given in an expert way), $TTend(\tilde{x}_i, \tilde{x}_j)$ – is the linguistic relation fixing the type of change between two evaluations $\tilde{x}_i$, $\tilde{x}_j$ of the scale, $RTend(\tilde{x}_i, \tilde{x}_j)$ – is the linguistic relation fixing the intensity of difference between two evaluations $\tilde{x}_i$, $\tilde{x}_j$ of the scale.

The relation $TTend(\tilde{x}_i, \tilde{x}_j)$ is the fuzzy linguistic relation which is applied for determination of comparative linguistic evaluation $\nu_{ij} = TTend(\tilde{x}_i, \tilde{x}_j)$ which characterizes the direction of change (increase or decrease) of the value of the absolute LE $\tilde{x}_i$ with respect to $\tilde{x}_j$ which can be represented by linguistic expressions, for example, by values from the set {INCREASE, DECREASE, STABILITY, ZERO}. Let us note that each evaluation $\nu_{ij} = TTend(\tilde{x}_i, \tilde{x}_j)$ is representable by its fuzzy set. The relation *TTend* is antireflexive, antisymmetric and transitive:

$$\forall \tilde{x} \in \tilde{X} \quad TTend\ (\tilde{x}, \tilde{x}) = 0,$$

$$\forall \tilde{x}, \tilde{y} \in \tilde{X}\ (\tilde{x} \neq \tilde{y}) \quad TTend\ (\tilde{x}, \tilde{y}) \wedge TTend\ (\tilde{y}, \tilde{x}) = 0$$

$$\forall \tilde{x}, \tilde{y}, \tilde{z} \in \tilde{X} \quad TTend\ (\tilde{x}, \tilde{z}) > TTend\ (\tilde{x}, \tilde{y}) \wedge TTend\ (\tilde{y}, \tilde{z})$$

The stated properties of the relation *TTend* permit to classify it as an ordering relation. Then the aggregate of all possible evaluations $V=\{\nu_{ij}\}$ forms the fuzzy ordinal scale $S_v = <Name\_TTend, V, \tilde{X}, G_v, P_v>$.

The relation $RTend(\tilde{x}_i, \tilde{x}_j)$ is also the fuzzy linguistic relation applied for determination of comparative linguistic evaluation $\alpha_{ij} = RTend(\tilde{x}_i, \tilde{x}_j)$ which characterizes the degree of difference, "non-metric" distance between $\tilde{x}_i$, $\tilde{x}_j$ which can be expressed linguistically, for example, by values from the set {BIG, MEDIUM, SMALL, ZERO}. This evaluation α_{ij} is also representable by its fuzzy set. The relation *RTend* is antireflexive and symmetric:

$$\forall \tilde{x} \in \tilde{X} \quad RTend\ (\tilde{x}, \tilde{x}) = 0,$$

$$\forall \tilde{x}, \tilde{y} \in \tilde{X} \quad RTend\ (\tilde{x}, \tilde{y}) = RTend\ (\tilde{y}, \tilde{x})$$

The indicated properties of the relation *RTend* permit to classify it as a relation of difference, with it the aggregate of all possible evaluations $A=\{\alpha_{ij}\}$ forms the fuzzy scale $S_a = <Name\_RTend, A, \tilde{X}, G_a, P_a>$.

Thus, the ACL-scale S_x for determination of linguistic evaluations is the two-level scale. At the first level of hierarchy from its universal set the ACL-scale S_x permits to determine linguistic evaluations $\tilde{x}_i$ for values $x \in X$ which characterize their qualitative aspects. Such linguistic evaluations relate to the class of absolute LE. At the second level of hierarchy for values $\tilde{x}_i$ and $\tilde{x}_j$ – linguistic evaluations of their changes (ν_{ij}, α_{ij}) which characterize qualitative aspects of differences or "difference of the first order" by scales S_v, S_a. Such linguistic evaluations are related to comparative LE.

Let us consider peculiarities of ACL-scales. The offered linguistic ACL-scale S_x is related to the class of fuzzy evaluation scales which enter into the class of ordinal scales, difference and the degree of difference can be additionally evaluated in it. This property permits to consider the linguistic evaluation ACL-scale S_x as quasi-interval and to determine "evaluation" and "computing" operations for it.

Let us introduce the following "evaluation" operations of the ACL-scale S_x generating linguistic evaluations:

1. The operation of determination of the absolute linguistic evaluation $\tilde{x}_i$ by the value of characteristic of the object x_j being evaluated
$$\tilde{x}_i = Fuzzy(x_j), x_j \in B, \tilde{x}_i \in \tilde{X}.$$
2. The operation of determination of the value of characteristic of the object x_j being evaluated by the absolute linguistic evaluation $\tilde{x}_i$
$$x_j = DeFuzzy(\tilde{x}_i), x_j \in B, \tilde{x}_i \in \tilde{X}.$$
3. The operation of determination of the type of difference (comparative linguistic evaluation)
$$\nu_{ij} = TTend(\tilde{x}_i, \tilde{x}_j), \tilde{x}_i \in \tilde{X}, \tilde{x}_j \in \tilde{X}.$$
The operation *TTend* is non-commutative.
4. The operation of determination of intensity of difference (comparative linguistic evaluation)
$$\alpha_{ij} = RTend(\tilde{x}_i, \tilde{x}_j), \tilde{x}_i \in \tilde{X}, \tilde{x}_j \in \tilde{X}.$$
The operation *RTend* is commutative.

Let us determine the aggregate of "computing" operations of the ACL-scale for generated linguistic evaluations:

1. Computing the absolute linguistic evaluation
$$\tilde{x}_j = Comp(\tilde{x}_i, \nu_{ij}, \alpha_{ij}).$$
2. The difference of intensities of differences
$$\alpha_{ij} = Diff(\alpha_i, \alpha_j)$$

3. The union of intensities of the difference
$$\alpha_{ij} = Union(\alpha_i, \alpha_j).$$
4. The intersection of intensities of differences
$$\alpha_{ij} = Inter(\alpha_i, \alpha_j).$$

Operations *Diff*, *Union*, *Inter* are commutative, associative, bounded.

Linguistic evaluations received by the indicated linguistic ACL-scale will be used in the next part as semantic-dependent for solution of the problem of summarization of FTS in terms of fuzzy tendencies.

IV. Application of Elementary Tendency Model for Summarization of Fuzzy Time Series

Let us consider the application of the offered ACL-scale in solution of the problem of summarization of FTS as the problem of identification of its general fuzzy tendency.

For this purpose, let us design the hierarchical granular model for the initial time series $X=\{t_i,x_i\}$, $i \in [1,n]$, n – is the quantity of members of the series. Let us introduce four levels of granulation forming of which corresponds to the bottom-up approach. The zero level of granules will be represented by fuzzy labels of the initial TS. For forming the granules of the zero level, let us use the "evaluation" operation of the ACL-scale: $\tilde{x}_i = Fuzzy(x_i)$.

Let us compare fuzzy tendencies of FTS and information granules which have structural commonness. Let us define the operation of granulation of the first level in the form of the functional *ETend* forming the granules of elementary fuzzy tendencies: $\tau i = ETend(\tilde{x}_i, \tilde{x}_{i+1})$, $i \in [1,n-1]$, n – is the quantity of members of FTS.

The functional *ETend* generates granules of elementary fuzzy tendencies on the basis of "evaluation" operations *TTend* and *RTend* of the ACL-scale and can be realized in the subsystem of fuzzy deduction with rules of the following form:

$R_1 :=$ IF X_t is A_{11} AND X_{t+1} is A_{12} THEN v_t is B_1 *AND* a_t *is* N_1

.....

$R_m :=$ IF X_t is A_{m1} AND X_{t+1} is A_{m2} THEN v_t is B_m AND a_t is N_m

The semantics of rules of realization of the functional *ETend* is represented in the following table.

TABLE I. TABLE OF RULES

No. of the rule	$\tilde{X}_t$	$\tilde{X}_{t+1}$	$v_t = TTend(\tilde{X}_t, \tilde{X}_{t+1})$	$\alpha_t = RTend(\tilde{X}_t, \tilde{X}_{t+1})$
1.	Sat	Sat	St	Ze
2.	Go	Go	St	Ze
3.	Ex	Ex	St	Ze
4.	Bad	Bad	St	Ze
5.	Ze	Ze	St	Ze
6.	Ze	Bad	Inc	Sm
7.	Ze	Sat	Inc	Me
8.	Ze	Go	Inc	Bi
9.	Ze	Ex	Inc	VeBi
10.	Bad	Ze	Dec	Sm
11.	Bad	Sat	Inc	Sm
12.	Bad	Go	Inc	Me
13.	Bad	Ex	Inc	Bi
14.	Sat	Ze	Dec	Me
15.	Sat	Bad	Dec	Sm
16.	Sat	Go	Inc	Sm
17.	Sat	Ex	Inc	Me
18.	Go	Ze	Dec	Bi
19.	Go	Bad	Dec	Me
20.	Go	Sat	Dec	Sm
21.	Go	Ex	Inc	Sm
22.	Ex	Ze	Dec	VeBi
23.	Ex	Bad	Dec	Bi
24.	Ex	Sat	Dec	Me
25.	Ex	Go	Dec	Sm

The following abridgements were used in this table: *Ze(Zero), Bad(Bad), Sat(Satisfactory), Go(Good), Ex(Excellent)* for fuzzy values of FTS; *Inc(INCREASE), Dec(DECREASE), St(STABILITY)* for values of types of changes; *Bi(Big), Me(Medium), Sm(Small) and modifiers, such as Ve(Very), Si(Significantly), No(No)* for values of intensity of changes.

Let us define the operation of granulation of the second level in the form of the functional *STend* forming the granules of local fuzzy tendencies: $\tau_j = STend(\tau_i, \tau_s)$, where τ_i, τ_s are granules of the first level.

The introduced functional *STend* is computed as the result of the union of one-type elementary tendencies on the base of the "computing" operation *Union* of the ACL-scale. Then the union $\tau_j = STend(\tau_i, \tau_s)$ is the such fuzzy tendency for which $v_j = v_i$, $\alpha_j = Union(\alpha_i, \alpha_s)$, $\mu_j = \mu_i \cup \mu_s$, the duration $\Delta t_j = \Delta t_i + \Delta t_s$.

The operation of union of one-type tendencies defines the granules of the second level.

The generalized form of rules of granulation of the second level on the basis of the functional *STend* has the form:

$R_{11} :=$ *IF* v_t *is Inc THEN*

IF a_t *is* A_{11} *AND* a_{t+1} *is* A_{12} *THEN TInc is* B_1

.....

$R_{m1} :=$ *IF* v_t *is Inc THEN*

IF a_t *is* A_{m1} *AND* a_{t+1} *is* A_{m2} *THEN TInc is* B_m ,

$R_{12} :=$ *IF* v_t *is Dec TH EN*

IF a_t *is* A_{11} *AND* a_{t+1} *is* A_{12} *THEN TDe c is* B_1

....

$R_{m2} :=$ *IF* v_t *is Dec THE N*

IF a_t *is* A_{m1} *AND* a_{t+1} *is* A_{m2} *THEN TDe c is* B_m

The semantics of rules of granulation of the second level is given below:

TABLE 2. TABLE OF RULES LEVEL 2

	TInc	*TDec*	*Tend*
1	Bi	Me	Inc
2	VeSm	Sm	Dec
4	Me	Bi	Dec
5	Bi	Sm	Inc
6	Me	Sm	Inc
7	Sm	Me	Dec
	….	…	…

On the basis of introduced functionals we defined the procedure of summarization of TS as the procedure of identification of the fuzzy tendency *Tend*. This procedure is the sequential generation of information granules which model TS at different abstract levels. The result of the procedure of summarization of FTS is the granule of the general fuzzy tendency which is the convolution of elementary tendencies into the linguistic evaluation of behaviour of a FTS:

$$ETend(\tilde{x}_i, \tilde{x}_{i+1}) \rightarrow STend(\tau_i, \tau_s) \rightarrow GTend(\tau_i, \tau_s).$$

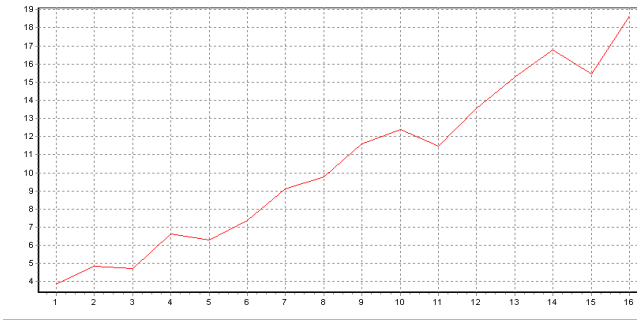


Figure 1. The time series No. 1. The result of summarization procedure: "General tendency =Increase"

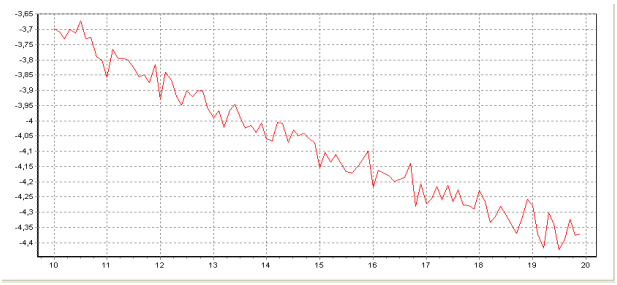


Figure 2. The time series No. 2. The result of summarization procedure "General tendency=Decrease".

The offered approach to the solution of the problem of summarization of FTS on the basis of fuzzy tendency model and granular computing was realized as software in the system generating artificial time series with noise. On figures 1, 2 examples of execution of the procedure of summarization of the artificial TS and its results are presented.

REFERENCES

[1] Batyrshinand L. Sheremetov. Perception Based Time Series Data Mining for Decision Making. /IFSA'07 Theoretical Advances and Applications of Fuzzy Logic, pp.209-219.

[2] N.G. Yarushkina Principles of the theory of fuzzy and hybrid systems: Training aid. - Moscow: Finances and statistics, 2004. - 320 p.

[3] M. Şah ,K.Y. Degtiarev. Forecasting Enrollment Model Based on First-Order Fuzzy Time Series. İnternational Conference on Computational İntelligence (İCCİ) 17-19 December 2004, İstanbul, Turkey

[4] Dvorak A.,Novak V.:Formal Theories and Linguistic Description. Fuzzy Sets and Systems, 143(2004), 169-188 .

[5] Song, Q., Chissom, B.S. Fuzzy time series and its models. Fuzzy Sets and Systems , 54 (1993) 269–277

[6] Huarng, K. Heuristic models of fuzzy time series for forecasting. Fuzzy Sets and Systems, 123: 369-386. Young, *The Technical Writers Handbook*, Mill Valley, CA: University Science, 1989.

[7] L.A. Zadeh. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. Fuzzy Sets and Systems , 90(1997), 111-127

Meta-Model and Platform for quickly build software applications

Yuri Rogozov, Alexey Degtyarev
System analysis and telecommunications.
Taganrog Institute of Technology, Southern Federal University
Taganrog, Russia
rogozov@tsure.ru, alexey.a.degtyarev@gmail.com

***Abstract*— In the development of large and complex applications is difficult to satisfy unique customer requirements without stretching development time. The need to support adaptability of applications is growing because of dynamic business processes. We analyzed of the literature in this domain. The analysis has shown that combination of the existing paradigm of application development is a good approach to support adaptability of business applications. This paper presents a multi-layered Meta-Model for enterprise applications and platforms that we have created on our multi-layered Meta-Model.**

Keywords - metamodel, metamodelling, meta-design, business application development platform, automation.

I. INTRODUCTION

The creation of business applications based on client-server architecture is still an urgent task. In this paper we will pay attention to the corporate business applications. The main features of such applications are heterogeneity (heterogeneity) of business information and variability processes that use this information. This creates a lot of problems in the development of business applications. Many of the studies on software project failures have identified difficulties of accurately capturing user requirements as a major contributor to failure of software development projects [1]. In the domain of software engineering a lot of effort has been spent to improve methods of capturing requirements. However, experience has shown that requirements cannot be initially captured completely and correctly [2]. In our opinion they are not static, but dynamically changing, evolving along with the world around us. To solve this problem software engineering proposed iterative approach to application development. The main idea of the approach is the speedy transfer of user working version [3]. Given the scale and complexity of corporate business applications, we can confidently say that even when using an iterative approach, the average time of development business applications is not less than six months. Approved by the functional and data structure in an early iteration of the project completion can be drastically revised by future system users. It will make some additional risks in the development process. That is why the support of business applications adaptability appears [27]. On the other side the market of the modern information technology, shows that universal automation solutions are ineffective [28], so the business application must be created individually for each customer [29].

To solve the above problems, we need such software tool that would allow: significantly reduce the complexity of each iteration; ensure the creation of individual decisions at different levels of detail (either total subsystem, or individual business functions); modify the functional and data structures at no additional cost, which were approved in early iterations.

Analysis of existing approaches to application development allowed us to determine that each of them focuses on a specific level of granularity, i.e. strictly defined in advance the items that are indivisible or atomic. In some approaches to such elements include operators of classical programming languages, classes and objects [4], in other cases, they include the whole program modules [5]. However, in our opinion, the approach Meta – Design is the most promising. [6]. A comparison of these approaches with its problems allowed us to assume that a multi-level Meta - Model platform was the most promising. To understand what must be the Meta - Model, we have generalized our knowledge in developing enterprise business applications. Then we built meta-level aspects of the model that described the business applications. This Meta-Model should be reflected in the Meta - Model implementation. We relied on our experience and knowledge and tried to make it more concrete. We chose the concept of service-oriented architecture as a starting point [7]. In this paper we propose a multi - level Meta - Model and platform for building client / server enterprise business applications.

Platform with developed AV Script scripting language and the unique structure of an independent data base [8] allow us to bind together the software details of different levels - from the software modules that implement support for business process or some of its functions to a low-level programming constructs perform mathematical operations and access to business information.

II. ANALYSIS OF EXISTING APPROACHES TO SOFTWARE BUILDING

During the development of software engineering a number of approaches to software development have formed. These approaches have both positive sides and restrictions. The most important methods of software development , proven during its existence presented below: a) Classical programming [9]; b) Generative programming [10]; c) Model Driven (Architecture, Design, Engineering) [11]; d) Feature-Driven

Development [12]; e) domain specific (languages ,modeling, design) [13]; f) Metaprogramming [14]; g) Software factory, Frame Technology [15].

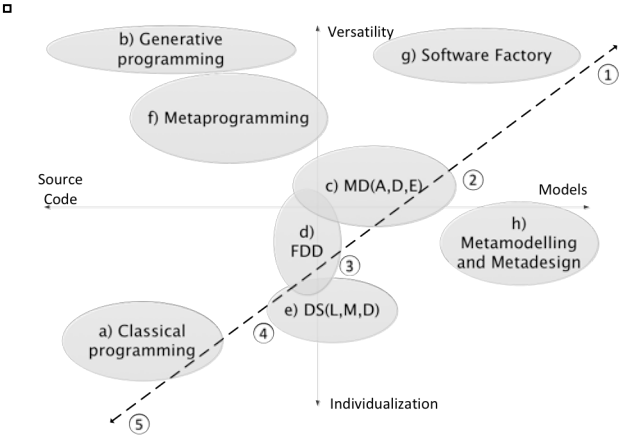


Figure 1. Existing approaches to software building

In our view, the criteria by which we can compare the approaches are following:

- level of personalization of the solutions determines how much depth we take into account the unique requirements of user (if the user wants a button, which shimmers like a rainbow, and we fulfill his requirement, it means that degree of personalization is high, if we make the four colors button only, because we have such templates, it means that degree of personalization is lower);
- level of formalization of the programming process, determines which way displaying the business problem in an existing space of solutions the process is carried out. (For example, if an application requires a creative activity for transformation of business models processes then the degree of formalization is low, and if an application has to perform a set of strictly defined rules for transformation of business models process then the degree of formalization is high) [16]. Then a comparison of approaches could be presented graphically in the following way (see Figure 1). The motion on the dotted line from left to right in Figure 1 shows the increasing abstraction level. So, to create a platform that would allow to reduce the complexity of each iteration, we need to maximize the formalization of the software development process.

Figure 1 shows that requirements to the platform could be met using Meta – design approach as the most attractive in accordance with our requirements [17].

However, the development of software at different levels of detail by constructing a common meta-model is most likely not succeeded. This is primarily because of constructing the complexity single universal Meta – Model [18, 19, 20]. Assumes that the Meta - Model should be multilevel and meet the following requirements: ensure that changes in the level of detail (see Figure 1); ensure the maximum degree of software formalization development processes.

III. ASPECTS THAT CHARACTERIZE THE CORPORATE BUSINESS APPLICATIONS

We believe that the modern corporate business applications aimed at collecting, storing and processing information in the most general case characterized by the following aspects (see Figure 2): work flow logic; graphic user interface; decision logic; data manipulating function; business objects. Note that some researchers in the software field have a similar view [21],[22],[23],[24], it makes our judgments more reasonable.

Work flow logic - an aspect that reflects the functions sequence performing, business applications and management tasks.

Graphic user interface - an aspect that reflects the interaction of users with information and business application.

Decision logic - an aspect that describes the implementation of logic solutions (calculation formulas, processing information about Business Objects).

Data manipulating function - determines how to work with data that represent real business objects. The standard functions for manipulating data include: the input new data, edit existing data, viewing of existing data, delete existing data.

Business Objects - a simplified model of real business objects, which characterizes them as the way which is important for building enterprise business applications.

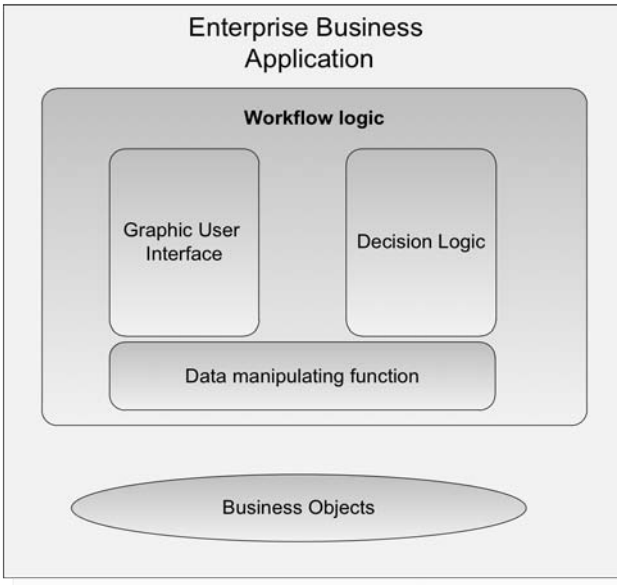


Figure 2. Aspects that characterize the corporate business applications

Figure 2 shows the Meta – Model aspects level that characterize the corporate business applications. This Meta - Model is abstracted from the domain and business objectives, i.e. is interdisciplinary. However, if we attempt to deepen it (refine) to a specific business problem we will be faced with problems. First, these problems are associated with a set of the

Identify applicable sponsor/s here. If no sponsors, delete this text box. (sponsors)

domain knowledge, which subject that works out in detail should know. Secondly knowledge of implementation technologies to create the functioning business applications is required. In other words, it is not enough to work out in detail aspects of the Meta - Model to the level of a specific business problem. It is necessary to display the specific business objectives for space implementation. Today it is a serious problem. Assume the following: concretized aspects of the Meta - Model elements, will find their place in any of the approaches to building software that depicted in Figure 1.

Then conceptually multi-layered meta-model space realizations can be represented as a plane that is divided into different levels of abstraction (see Figure 3). In this view, you can find the correspondence between the abstraction level above approaches and input levels of abstraction space realizations (see numerals in Figures 2 and 3). Depth of detail and level of abstraction is limited only by the degree of our knowledge and technology developments i.e. can grow indefinitely.

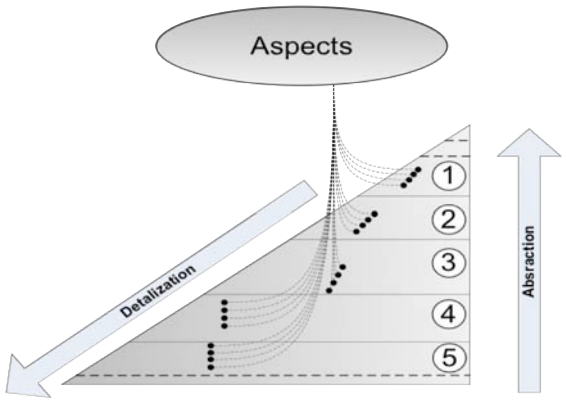


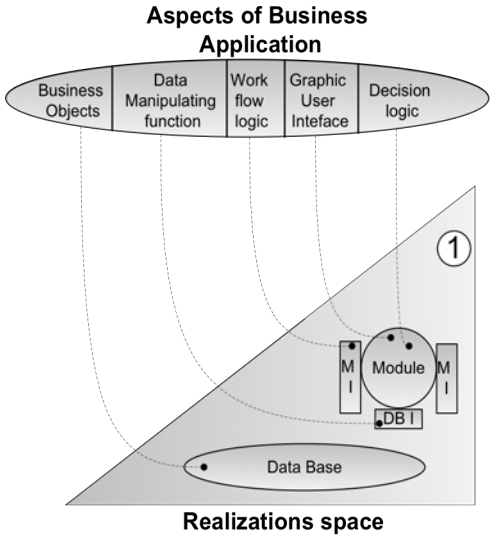
Figure 3. Mapping aspects

IV. MULTI-LEVEL META - MODEL CORPORATE BUSINESS APPLICATIONS

Multi-level Meta- Model was constructing by using the idea behind meta-approach of building prototypes of the object [25]. To determine the highest level of abstraction layered Meta- Model we used the principle of service-oriented architecture [7]. This principle bases that any enterprise business application is represented as a collection of loosely coupled modules. At the level of detail, where the module is an atomic unit, a business application can be represented as follows - Figure 4. By analogy with the terminology of object-oriented approach [26], the module is an object that encapsulates a user interface and business logic. Inter-module interfaces and interfaces to the database are external interfaces of the module. In Figure 4, weak ties are reflected by dotted lines. In accordance with Figure 3, consider how aspects of enterprise business applications are displayed at the proposed area of implementation. Analyzing the figures 2, 3 and 4 set a direct mapping is characterized by the corporate business applications in the proposed space realizations.

Figure 4. Meta-Model on level modules

Business objects are displayed in the database (Data Base), the logic of the process of business applications in the inter-module interfaces (Module interface), user interfaces and



business logic in the module (Module), the standard functions for manipulating data in an interface to the database (data base interface). This map is shown in Figure 5. Displays the next level of detail will be similar.

Figure 5. Mapping aspects of one level

Let's decompose the module into smaller components (see Figure 6) and introduce the corresponding concepts.

Module - is a certain amount of functional units (FU). They are strongly connected (tightly coupled) with each other and have minimal connection with the functional units (FU) of other modules. One of the functional units have to be the main (working with the central or main object in the context of the business objectives of the module) and the remaining subsidiaries.

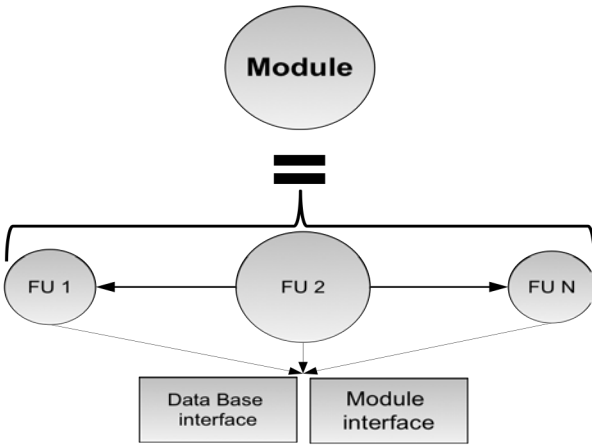


Figure 6. Meta-Model on level functional units

We showed Interface intermodule interactions (Module Interface) interface and work with DB (Data Base interface) on Figure 6 because the functional units themselves may interact with the database and functional units of other modules. In reality, the module interface and data base interface are located in the internal structure of functional units (FU). The solid lines in Figure 6 show that the functional units are combined into modules according to the principle of tightly coupled. We introduce the concept of functional units and continue to decompose.

Functional unit of the module (FE) - a graphical element (shape, window, Web page, etc.) which contains a set of elementary operations (EO). They are intended for display, modify, add or remove a business object's attributes or their values. Elementary operation may cause, another elementary operation or a functional unit of the module and perform other similar operations. In other words, a functional unit allows for a certain set of functions associated business processes.

Main functional unit (MFU) - functional unit of the module with the highest degree of connectivity (informational or functional) with other functional units of modules and comprises a master form. The remaining functional units of the module are subject-forms. We can lead explorer in MS Windows as an illustration. It folders and files are central to the business objects, so a window displaying them with all its functionality, is the primary functional unit. In turn, the window control which users can share any file or folder it can be a child or a subordinate.

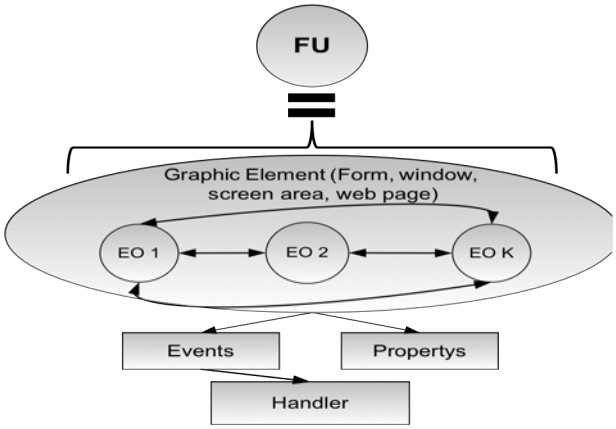


Figure 7. Meta-Model on level Graphic Element

The graphic element of a higher order (see Figure 7) - is a component, which is presented graphically in the window form or the screen. It has events and properties and contain a set of elementary operations. Events and properties of the graphical elements of higher order and a set of elementary operations together provide a complete description of the functional units of the module. Elementary operations can be arbitrarily interact with each other and interact with other functional units. There are two important limitations: an elementary operation cannot directly interact with the unit belonging operation to another graphic element, a functional unit of the module always has one and only one graphic element of a higher order, which represents a box shape of the screen or Web page.

Events - this is actually class methods (in the terminology of object-oriented programming) implements a specific graphic element (such as a mouse click - OnButtonClick (), or loss of window focus - FocusOff (), or closing the window - OnClose ()), each event if so stipulated requirements for business applications, can have its own handler.

Properties - the attributes that characterize a particular instance of a graphical interface. The properties of the graphic element may include the following parameters: name, description, the form position the top left corner along the axis X; the form position of the upper left corner along the axis Y; height form, the breadth of forms; color, font settings (font name, font size, bold, italic, underline), the visibility of the form (visible / invisible).

Elementary operation (EO) - is an elementary graphical elements (buttons, panels, switch, etc.), that connected with a single action function of a business process. The structure of its metadata elementary graphic element is very similar to the graphic element of a higher order. There is one exception, to be exact an elementary graphic element cannot contain elementary operations. Everything else is absolutely equally: events, properties and handlers. Let`s consider the structure of the handler (see Figure 8). The structure is identical for both the graphic element of a higher order (GE) and for an elementary graphic element (EGE).

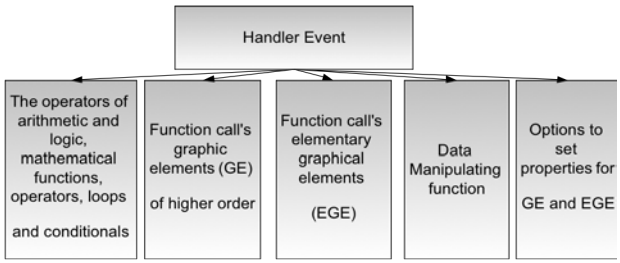


Figure 8. Meta-Model on level handler event

The operators of arithmetic and logic, mathematical functions, operators, loops and conditionals - these operators

are designed to implement the business logic (see Figure 1), their collection may vary.

Function call's graphic elements (GE) of higher order, function call's elementary graphical elements (EGE), options to set properties for GE and EGE - may represent a limited number of universal adaptive functions (performance depends on the method of implementation and using the framework).

Data Manipulating function - designed to implement interfaces with databases. Data Manipulating function can be variable. For stored objects in our database ("Object", "object attributes", "The values of object attributes") we released 12 basic functions for manipulating data. In case of need this set of functions can be expanded. The handler must have a certain syntax. The handler`s syntax can be a modern graphical syntax, built on the cognitive perception of reality as well as the classic text syntax.

A result of realizations stepwise decomposition of elements a Meta - Model was obtained. The Meta - Model has a divergent level of detail. Multi-level meta-atomic elements at the deepest level of detail can be considered: the elements of an event handler, basic graphic elements and graphic elements of the highest order. The last two sections cannot be divided but can be customized. Generalized multi-layered meta-model is shown on Figure 9.

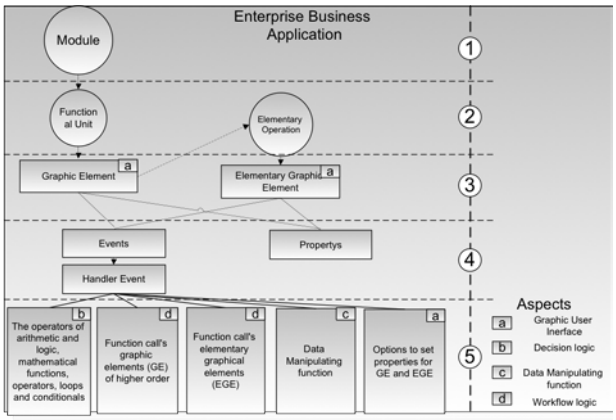


Figure 9. Meta-Model on all levels

We should note that at all levels either explicitly or implicitly event element as part of an integration is used. The syntax of the handler describes the event.

V. A PLATFORM TO QUICKLY BUILD BUSINESS APPLICATIONS

Our platform is based on the multi-level Meta – Model. The Meta - Model presented above has been implemented as bundles of desktop applications and server. Server stores the description of the components, business information and all service information. We tried to make the most simple quickly creation environment of business applications. It contains only the most necessary elements for building business applications. We will see a concise workspace, when we launch Primius platform. Since launching platform Primius, we will see a concise workspace. Main navigation menu consists of 2 major categories: Administration and modules that are available in selected business applications (see Figure 10). Figure 10 shows that the current business application consists of 3 modules - Drugstore, Clinical nutrition, Employee.

The modules are workstations of employees in the enterprise information system. Each application contains several functional units. The work of the existing modules is no differ from other business applications that is why it will not be considered. Figure 11 shows the category of platform administration. This section consists of three sections: Configuration, System Editor, and Business Object Editor. The Configuration partition partially consistent with the hierarchy of levels of multilevel Meta - Model we have developed. The creation and configuration the necessary business applications are made by this section. Section System Editor is designed to create new and modify existing items that are available in the categories section of Configuration. The work of Configuration partition always starts with the choice of Domain. Any further action should be tied to a specific application, so long as the current application is not specified, the other tabs will be unavailable.

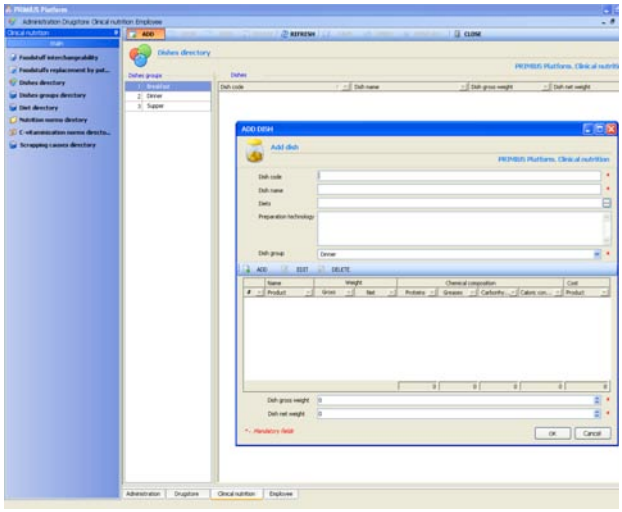


Figure 10. Modules those are available in selected business applications

A configuration on the modules is done by adding them to the list of available in the selected application modules. Every time you add a new module to the application, the metadata of the business data changes. Features of the implementation of this framework were considered in [8]. Configurations of next detail levels of elements realizes similarly. Figure 12 shows the Business Objects Editor. Any business object (real or abstract) is displayed by directory of objects that reference the attributes and the type of relationship between objects and attributes. All business objects and their attributes are linked to a specific system. There are two ways to create or change the structure of business data: automatic and manual.

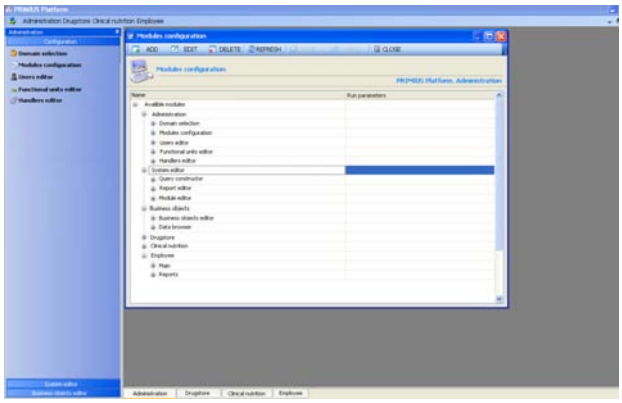


Figure 11. Modules configuration

Automatic mode is available when developer operating at the level of modules. In other cases, the structure of business data should be prepared manually by editing the business objects.

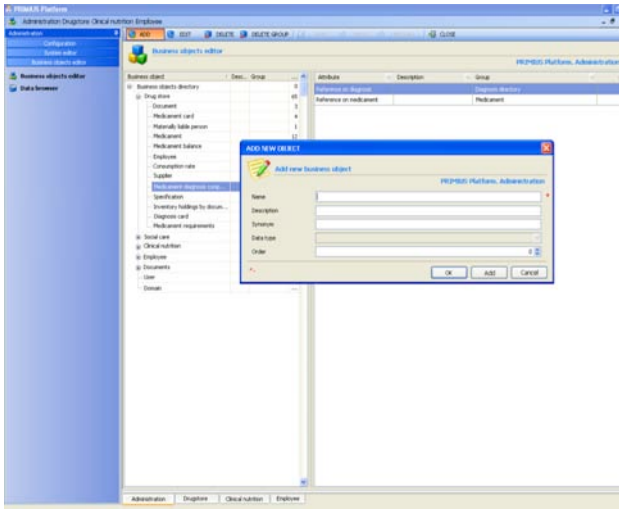


Figure 12. Business object editor

VI. CONCLUSION

In this article we looked at popular approaches to development programs and analyzed them for the important criteria to us. We concluded that to solve the problems of software development which relate to development time, individuality and flexibility of these solutions the platform should use a combination of approaches Meta design and factory applications. We have also assumed that the basis of the platform should be based on a multi-level Meta - Model. Analysis of the aspects characterizing the corporate business applications enabling us to understand what kind of needs multi-level Meta - Model should be. Our platform is based on the Meta - Model that we have developed. We have developed a unique structure of the data [8] to implement the ideas embodied in the Meta - Model. The structure is a common repository that describes elements of Meta, Meta data describing the business objects and data from which these objects work. This platform allows reducing the demands on staff. To explore this indicator we have attracted students to develop real-world business applications. Students were at 2 and 3 year bachelor's degree programs in "Computer Science and Engineering." One of the groups studied the C # and SQL in the IDE Microsoft Visual Studio 2005, another group of students studying specialized algorithmic language AV Scripts, SQL and basic knowledge of operating principles in the platform Primius. Our research group specializes in the development of fairly large business applications for the social protection institutions. We detected the following fact: in both groups to obtain basic knowledge took on average a similar time. However, the results between the groups were very different. Figure 13 reflects the evaluation of professional commitment to the development of applied business applications of groups.

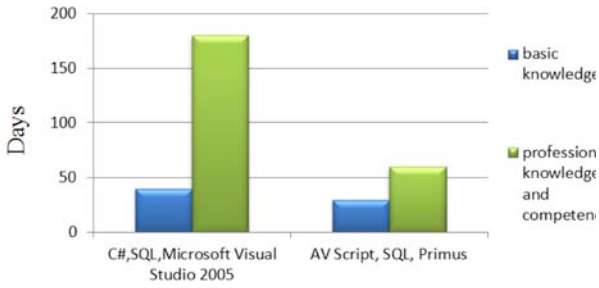


Figure 13. Comparison Chart

When we research and test platform for the control groups we identified areas which require further investigation. One of the identified areas was the task of generation automating the SQL queries. We want to rid the developers using our platform from having knowledge of SQL - they need to concentrate on the business structure of the data but not on relational algebra. This work demonstrates that a natural way of development of software engineering is raising the level of abstraction. Our results show that the combination paradigm of Meta - Design and application factories can produce very good results.

REFERENCES

[1] Standish_Group (1995). Chaos. THE STANDISH GROUP REPORT.

[2] Tao Yue, Lionel C. Briand and Yvan Labiche, "A systematic review of transformation approaches between user requirements and analysis models". //Springerlink.com

[3] Craig Larman, Victor R. Basili (June 2003). "Iterative and Incremental Development: A Brief History"

[4] Gafter, Neal (2006-11-05). "Reified Generkics for Java"

[5] Esteves, J., and Pastor, J., Enterprise Resource Planning Systems Research: An Annotated Bibliography, Communications of AIS, 7(8) pp. 2-54

[6] Cesar Gonzalez-Perez, Brian Henderson-Sellers Metamodelling for Software Engineering, Wiley, 2008, 219 pages

[7] Bieberstein et al., Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap (The developerWorks Series) (Hardcover), IBM Press books, 2005.

[8] Youri I. Rogozov, Alexander S. Sviridov, Sergey A. Kutcherov, Wladimir Bodrov. Purpose-driven approach for flexible structure-independent database design. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, Volume 1, p.356-362

[9] Stroustrup, Bjarne (1997). "1". The C++ Programming Language (Third ed.).

[10] Krzysztof Czarnecki and Ulrich W. Eisenecker Generative Programming - Methods, Tools, and Applications Addison-Wesley, 2000, 864 pages

[11] Anneke Kleppe , Jos Warmer , Wim Bast MDA Explained: The Model Driven Architecture(TM): Practice and Promise Addison-Wesley Professional, 2003, 192 pages

[12] Palmer, S.R., & Felsing, J.M. (2002). A Practical Guide to Feature-Driven Development. Prentice Hall. (ISBN 0-13-067615-2)

[13] Steven Kelly, Juha-Pekka Tolvanen Domain-Specific Modeling Wiley-IEEE Computer Society Press,2008, 448 pages

[14] David Abrahams, Aleksey Gurtovoy C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond. Addison-Wesley Professional , 2004, 4000 pages

[15] Jack Greenfield and Keith Short Software Factories: Assembling Applications with Patterns, Frameworks, Models & Tools Wiley, 2004, 500 pages

[16] Рогозов Ю.И., Актуальные проблемы и перспективные направления в области построения информационных систем и процессов: сборник статей международной научно-технической конференции. Таганрог: изд-во ТИ ЮФУ, 2010 г., стр. 9-15

[17] Ye, Y., Fischer, G.: Designing for Participation in Socio-Technical Software Systems. In: Stephanidis, C. (ed.) Proceedings of 4th International Conference on Universal Access in Human-Computer Interaction, Beijing, China, pp. 312–321. Springer, Heidelberg (2007)

[18] Schwabe, D., G. Rossi, et al. (1996). Systematic hypermedia application design with OOHDM. seventh ACM conference on Hypertext, Bethesda, Maryland, United States, ACM Press.

[19] Fratenali, P. and P. Paolini (1998). A conceptual model and a tool environment for developing more scalable and dynamic Web applications. EDBT 98, Valencia, Spain

[20] Schewe, K.-D., B. Thalheim, et al. (2004). Modelling and Stories in Web Information System. Information Systems Technology and its Applications (ISTA), Salt Lake Ciy, Utah, USA.

[21] Visual Rules – www.visual-rules.com/dynamic-applications.html

[22] Athula Ginige. Meta-design paradigm based approach for iterative rapid development of enterprise WEB applications. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, p.337-343

[23] Webratio – http://www.webratio.com

[24] Mendix – http://www.mendix.com/

[25] Yuri Rogozov, Wladimir Bodrow, META-APPROACH FOR CREATION OF OBJECT PROTOTYPES. Proceedings of ICERI2010 Conference. 15th-17th November 2010, Madrid, Spain.

[26] Grady Booch, Object-Oriented Analysis and Design with Applications, 2007.

[27] L.N. Lyadova et al., Implementation of distant learning portals based on CASE-technology METAS, // International Journal "Information Technologies and Knowledge", 2008. Т. 2. № 5. С. 489—495

[28] L.N. Lyadova, V.Lanin, Documents Management in Dynamically Adaptable Systems Based on Metamodelling // Proceedings of the Congress on Intelligence Systems and Technologies “AIS-IT’10”. Scientific publications in 4 volumes. Moscow: Physmathlit, 2010, Vol. 4. , Moscow, 2010.

[29] Fischer, G. and E. Giaccardi. Meta Design: A framework for the future of end user development. End User Development: Empowering People to flexibly Employ Advanced Information and Communication Technology. H. Lieberman, F. Paterno and V. Wulf, Springer. 9: 427-457

Application of SADT for source code generation in learning the programming fundamentals

Kustov M.[1], Guban B., Datsun N.[2]

Department of Applied Mathematics and Informatics
Donetsk National Technical University
Donetsk, Ukraine
[1]maksym.kustov@gmail.com, [2]datsun@pmi.dgtu.donetsk.ua

***Abstract*— Approach to generating source code from the SADT (Structured Analysis and Design Technique) specification of the program is offered. Invariants are allocated in basic algorithms. Algorithm of generating source code on the basis of templates is formulated. Data structures used for implementation of algorithm are considered. Internal representation of the SADT specification can be used to analyze the properties of the generated program.**

programming fundamentals, SADT, algorithms, invariants, data structures, generating source code

I. INTRODUCTION

This paper presents the results of application of the SADT (Structured Analysis and Design Technique) [1, 2] specification for the generating source code. The course SE101 of SEEK (Software Engineering Education Knowledge) [3] is focused on the solution of such learning objectives: develop simple statements of requirements and write small programs in some language. Modern methods of generating source code from a Domain-Specific Language (DSL)/specification language (SL) can be used in learning to create a structured program based on the standard algorithms and simple data structures. Development of methods for generating source code from a DSL/SL is an actual problem for the various subject areas [4-10]. There are textual and visual DSL/SL. UML and UML-like languages are most popular among the visual specification languages. C++, C#, Java, VB and VB.NET in most cases are the source code languages [4-9]. But the notation of UML and object-oriented languages for generating source code are redundant for problems of learning the programming fundamentals. SADT [1, 2] as a methodology for functional modeling successfully works for the description of clearly specified processes [11, 12]. Therefore it is possible to apply SADT in learning the methods of decomposition programs. Language SADT-methodology is a visual specification language. Graphical notation of IDEF0 [2] in such subject area as formalization of programs development allows to consider the logic relations between functional blocks of the program and provides the minimum toolkit for building software projects with standard algorithms. Work objective is to use the graphic notation of SADT for teaching of decomposition program and generating source code for procedural language from the SADT-specification with standard algorithms on basic data structures.

II. SELECTION OF INVARIANTS OF BASIC ALGORITHMS

A. Selection of Invariants of The Algorithm With Respect To The Input Data

Let's try to identify the invariants for basic search algorithm of the maximum negative elements. For this we consider the implementation of this algorithm for various data structures. Fig. 1 presents these algorithms for the array, a text file and linked list.

```
f=0;                    f=0;                    f=0;
i=0;                    fseek(file, 0L,         i=p;
                        SEEK_SET);              // p - head list
                        fscanf("%d",&a);
while(i<n)              while(!feof(file))      while(i!=NULL)
{                       {                       {
 a=x[i];                                         a=i->info;
 if((a<0)&&!f))          if(a<0&&!f)             if((a<0)&&!f)
   {                       {                       {
    f=1;                    f=1;                    f=1;
    res=a;                  res=a;                  res=a;
   }                       }                       }
 if(f&&(a<0)             if(f&&(a<0)             if(f&&(a<0)
    && (res<a))             &&(res<a))              &&(res<a))
       res=a;                  res=a;                  res=a;
   i++;                                            i=i->next;
}                       }                       }
if(!f)                  if(!f)                  if(!f)
{                       {                       {
 printf("There are      printf("There are       printf("There are
no right elements");    no right elements       no right elements
}                       ");                     ");
                        }                       }
 Algorithm A1            Algorithm B1            Algorithm C1
```

Figure 1. Basic algorithms of the maximum negative element for the array, a text file and linked list.

After detailed analysis of algorithms, you may notice that some parts are the same for all types of data structures. Let's distinguish immutable parts (invariants) of algorithms.

In Fig. 1, the invariants of these algorithms are marked in bold. Thus we have the invariants of algorithms with respect to the input data. That is, when you change structure of input data invariant remains the same. The resulting invariants are divided into three parts: initialization (in Fig. 1 marked in underlined), main part, ending part (in Fig. 1 marked in italic).

Unchanging part of the processing algorithm determines the input variable and the variable part of it represents this variable to the input, looking over an array, file or linked list.

B. Selection of the algorithm invariant by the method of processing

We consider algorithms for finding maximum of the negative elements, minimum and replacement of the positive elements of zeros for the same data structure (array). Fig. 2 presents these algorithms.

f=0;
i=0;
while(i<n)
{
a=x[i];
if((a<0)&&!f)
{
f=1;
res=a;
}
if(f&&(a<0)&&
(res<a))
res=a;
i++;
}
if(!f)
{
printf("There are
no right elements");
}
	res=x[0];
	i=0;
	while(i<n)
	{
	a=x[i];
	
	if(a<rez)
	
	res=a;
	i++;
	}
		i=0;
		while(i<n)
		{
		a=x[i];
		
		if(a>0)
		
		x[i]=0;
		i++;
		}

Algorithm A2	Algorithm B2	Algorithm C2

Figure 2. Various algorithms for the array.

Let's distinguish the invariants for the considered algorithms. Common to these algorithms is the cycle for the elements of the array. In Fig. 2, the invariants of these algorithms are marked in bold. In these algorithms there are no some parts: algorithm B2 has lost the ending part, and the algorithm C2 lost the initialization and the ending parts.

III. GENERATION OF NEW ALGORITHMS BASED ON INVARIANTS

Let's create a patterns of the algorithms obtained in the previous sections. To do this, we will cut out the invariants from the algorithm. Now we can get a brand new algorithm by combining the algorithm C1 and algorithm B2. Fig. 3 illustrates the preparation of an entirely new algorithm based on invariants. Further, the project implementation software for generating program code based on the specifications of the SADT-methodology will be discussed.

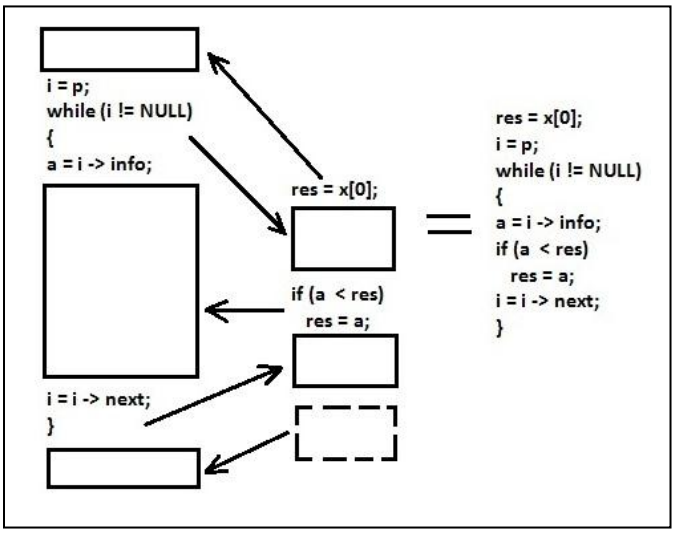


Figure 3. Process of obtaining a new algorithm based on invariants.

IV. DATA STRUCTURES

From the IDEF0/SADT graphic language for creation of the SADT specification are used:

- functional blocks (Activity Box) to denote the program modules at decomposition
- interface arcs (Arrow) of two types (“Input” to transfer the input data and “Output” to transfer the output data);
- names of interface arcs (Arrow Label) to denote the names of the input and output data.

Consider the representation of the visual components of SADT specification in the form of data structures

A. SADTUnit Structure

The main data structures is SADTUnit, describing a unit of SADT diagrams, and SADTData, describing the type and data structure of links SADT diagrams. Elements of structure SADTUnit are:

- number of diagram unit (integer);
- links (list);
- containers (list).

“Number of diagram unit” serves for process ordering. Process occurs consistently, according to numbers of diagrams.

List item "Links" is a pair <key (name); value (link structure)>. This component will organize a logical relationship of component with other diagram units.

List item "Containers" is a pair <key (name); value (container structure)>. It contains a set of possible patterns of code for generated program. Needed container is selected in accordance with the data type and structure of the input data.

B. SADTLink Structure

Elements of structure SADTLink are:

- name (string);
- type of link (string);
- linked to the diagram (string);
- link on the external diagram (string);
- data (string).

"Name" is used to display in graphical representation of the diagram. Also the name is used for this purpose what to refer to this communication from external diagrams and communication with various internal structures, such as the mask of using of the diagram container.

"Type of link" specifies the type of communication with respect to the diagram in which this connection is contained. Link can be:

- input (intended to link a diagram to input data);
- output (intended to create of the reference to the data received as a result of work of the diagram);
- modular (intended to connect the various modules to the diagram, which can extend standard functionality).

"Linked to the diagram" specifies the name of an external diagram. It is used in input and modular links to specify the diagram connected with the current diagram through this link.

"Link on the external diagram" is the name of the output link of external diagram, which is connected with this link. It is used for input and modular links.

The field "Data" is used for output links. The value of this field corresponds to a name of data which are created as a result of work of the current unit diagram. This field establishes compliance between data and links.

If the diagram generates new data, the fields "Linked to the diagram" and "Link on the external diagram" are empty, and the field "Data" refers to the structure of the newly created data. Such a set is created only for the output links. Conversely, if the field "Data" is empty, and the fields "Linked to the diagram" and "Link on the external diagram" are filled, it means that the link is connected with one of the external diagrams, or that the link is associated with one of the inputs to the current diagram. This occurs when new data aren't generated, but only only to modify the input data.

C. SADTData Structure

Elements of structure SADTData are:

- data structure (string);
- data type (string);
- name of the data (string).

"Data structure" describes the type of the data structure (array, list, file).

"Data type" describes the type of data in structure (integer, character or real).

"Name of the data" used to identify the created data structure.

By analogy to SADT diagrams, SADTUnit itself acts as a diagram, and SADTData acts as links between diagrams. If we consider the mechanism of processing of the workpiece in the SADT specification, SADTData will act as a workpiece, and SADTUnit will act as methods and techniques for handling the workpiece. Accordingly, after processing of the workpiece is obtained by the final product, which will be the output link. It will be at the same time presented by SADTData structure.

D. Container Structure

Elements of container structure are:

- text field (string);
- mask of using (array of structures).

"Text field" contains one of versions of the generated text of the program.

Structure of the field "Mask of using" contain the name, structure and type of input links. The container contains some such structures. If all parameters (structure and type), specified in the mask, correspond to all the listed links, this particular container is used for source code generation.

E. Mask of Using Structure

The mask of using is the array, each element of which is the data structure:

- name of link (string);
- data type (string);
- data structure (string).

“Name of link” should correspond to one of names of links of diagram units.

Type in the "Data Type" should match the type of the corresponding input link. Then the container will be called, and the code will be generated by its pattern.

"Data Structure" contains the name of the data structure relevant to the input link.

Compliance of a mask of the input link is defined by the coincidence of the input link name of diagram and the name of link in the mask of using.

F. Data Structures Communication

The main block is «SADTUnit». It contains the array of structures «Container» and «SADTLink». In turn, the structure of the container contains the array of of structures «Mask of Using», and the field «Data» of structure «SADTLink» refers to a structure «SADTData».

V. SOURCE CODE GENERATION

A. Description of Problem

Input data:

ST: set /* set of basic data structures */
ST={array, matrix, list, sequential file, direct access file}
AT: set /* set of basic algorithms */
BD: set /* set of diagram blocks */
BD = <nbdid, BDIN, BDOUT, MET, INST>
nbdid: string /* ID of the diagram block */
BDIN: set /* set of inputs of diagram */
BDOUT: set /* set of outputs of diagram */
BDMET: set /* set of diagram methods*/
INST: set /* set of tools */

Restrictions: AT, ST, BD sets are not empty.

Results:

D: graph /* the problem specification from which the file PR is generated */
PR: file /* file of program code */

Relation " input data - results ":

D=BD union V
$V = \{v_i\}$
$v_i = < bdin^k_i , bdout^l_j >$, i, j — numbers of input and output block diagram, k, l - numbers of inputs and outputs
$PR = AT \times ST \times P$

B. Source Code Generation by SADT Specification

Using of templates is a common practice in programming. Many programming languages have functions to work with templates. But the usual patterns become powerless when the variables substituted into the templates have different structure (arrays, lists, files), because the processing of each data structures is individual. For the array, it is possible to use the loop, the matrix is required nested loops, for a tree - bypassing the depth or breadth.

Several containers of a template used to solve this problem. Appropriate container is selected according to the type and structure of the input data. After that contents of the container are used as a usual template.

Code generation is a substitution of data from the input links in the text of container. The universality of this approach is that the containers are defined for each data type and data structure.

C. Source Code Generation Algorithm

1. Creation SADT diagrams.

2. Creation of links for SADT diagrams and giving them names.

3. Numbering units in order to code generate.

4. Creation of containers with templates.

5. Specifying of mask of using for each of containers. The algorithms contained in the containers can be created on the basis of invariants of the algorithms.

6. Linkage of the created blocks. Linking of blocks occurs by comparison to the input link of one diagram and the output links of another diagram. Fields «Linked to the diagram» and «Link on the external diagram» are filled. Link will be recursively traced to the reference to data.

7. Selection of the container, which corresponds to the input data. For this purpose it is necessary to compare the data of each input link. Then, knowing type and data structure with which should operate the diagram block, select the appropriate template. From the list of all possible patterns is chosen the one with the mask of using corresponds to the input data.

8. Replacement in a template of the container of values of input links on names of data which correspond to this link.

9. Repeating steps 1-8 to all diagrams.

10. Arrangement of the obtained text of a program code according to numbers of diagrams.

VI. DESCRIPTION OF TEST STAND

A. Implementation Toolkit

Qt library [13] was chosen as an instrument of implementation. It includes all the basic classes that may be required for the development of applied software. Qt is fully object-oriented tool, easily expanded and supporting of component programming technique. Thus, the designed software product is cross-platform and can easily extended to work over the network and databases.

Development of software is based on the Linux operating system CentOS 5.5. Tests were carried out for the following operating systems CentOS5.5, Fedora 14, Windows XP. Compilation and debugging implemented in QT Creator.

B. Architecture Variants

Experiments were carried out on different hardware architectures running different operating systems. Table I presents options for the test stand.

TABLE I. VARIANTS OF TEST STAND

Hardware	OS
Intel(R) Celeron(R) CPU E3300 2.5GHz 1GB RAM	Microsoft Windows XP Professional Service Pack 3
AMD Phenom(tm) II X4 B45 Processor 3,1 GHz 1370112 KB RAM	CentOS 5.5 x86_64
AMD Sempron(tm) 140 Processor 2,7GHz 1796440 KB RAM	Fedora 14 x86_64

VII. TESTING

A. Description of Test Set

The experiment was performed on data structures: the array of integers, the direct access file of integers, the array of strings.

The blocks of diagram with containers ("find the minimum", "find the maximum", "swap elements") were created for processing of these data structures.

Containers are contained in each of blocks of diagrams. Each of the container corresponds to a certain set of input data. Table II presents the set of containers for each of the block.

TABLE II. SETS CONTAINERS FOR EACH BLOCK

Data	Container
Array of Integer	
Один контейнер, создающий данные	int an=20; int a[20];
Direct Access File of Integers (D/A File Int)	

Один контейнер, создающий данные	FILE *f;
	if((f=fopen("rw","filename"))==NULL)
	printf("Cannot open file");
	int fn=fseek(a,0,SEEK_END);

Array of Strings	
Один контейнер, создающий данные	int cn=20; char c[20][100];
Minimum in Dataset (Min in Dataset)	

ArrayInt	int iMin=0;
	for (int i=1; i<%xn%;i++) {
	if(%x%[i]<%x%[iMin]) iMin=i;
	}

DAFileInt	int iMin=0;
	int Min,bufMin;
	fseek(%x%,0,SEEK_SET);
	fread (&Min, sizeof(Min), 1, %x%);
	for (int i=1; i<%xn%;i++)
	{
	fread (&bufMin, sizeof(Min), 1, %x%);
	if(bufMin<Min)
	{ Min=bufMin; iMin=i; }
	}

ArrayString	int iMin=0;
	for (int i=1; i<%xn%;i++)
	{
	if(strcmp(%x%[i],%x%[iMin])<0) iMin=i;
	}

Maximum in Dataset (Max in Dataset)	
ArrayInt, DAFileInt, ArrayString	By analogy to the container Min in Dataset
Swap elements in Dataset	
ArrayInt	int buf; buf=%x%[%i1%]); %x%[%i1%]=%x%[%i2%]; %x%[%i2%]=buf;

DAFileInt	int changeMax,changeMin;
	fseek(%x%,iMin*sizeof(int),SEEK_SET);
	fread (&changeMin, sizeof(Max), 1, %x%);
	fseek(%x%,iMax*sizeof(int),SEEK_SET);
	fread (&changeMax, sizeof(Max), 1, %x%);
	fseek(%x%,iMin*sizeof(int),SEEK_SET);
	fwrite (&changeMax, sizeof(Max), 1, %x%);
	fseek(%x%,iMax*sizeof(int),SEEK_SET);
	fwrite (&changeMin, sizeof(Max), 1, %x%);

ArrayString	char buf[100]; strcpy(buf,%x%[%i1%]);
	strcpy(%x%[%i1%],%x%[%i2%]);
	strcpy(%x%[%i2%],buf);

Fig. 4 shows the interface of the test stand.

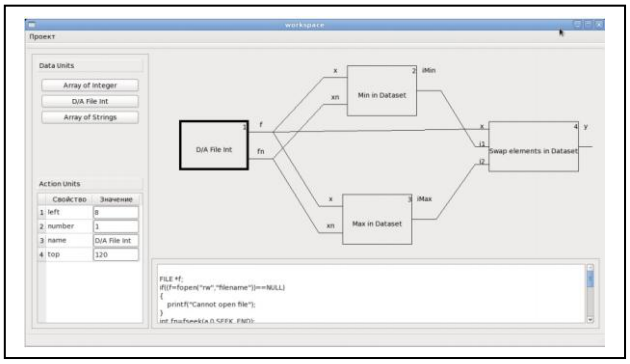


Figure 4. Interface of the test stand.

VIII. EXPANSION OF SET OF BASIC ALGORITHMS

Let's consider source code generation by the SADT specification on an example of search of an element "a" an and further removal of the found element from the one-dimensional array and the list. In Fig. 5, the invariants of these algorithms are marked in bold.

f=0;
res=-1;
i=0;
while ((i<n) && !f)
{
r=x[i];
if (r==a) {
f=1;
res=i; }
else
i++;
}
Algorithm A3
	f=0;
	res=-1;
	i=p;
	while ((i!=NULL) && !f)
	{
	r=i->info;
	if (r==a) {
	f=1;
	res=i; }
	else
	i=i->next;
	}
	Algorithm B3

Figure 5. Element search (array and list).

Having analyzed algorithms of A3-B3 and A4-B4, we see that some parts of these algorithms are identical. Fig. 6 shows that the same algorithm is specific to the different data structures.

i=0;
while (i<Res)
i++;
while (i<n)
{
a[i]=a[i+1];
i++;
}
Algorithm A4
	i=p; // p - head list
	while (i!=Res)
	i=i->next;
	while (i!=NULL)
	{
	i->info=i->next->info;
	i=i->next;
	}
	Algorithm B4

Figure 6. Remove with a shift (array and list).

Fig. 7 represents the result of combining these two basic algorithms. Under the proposed approach in this work, it is possible to allocate the invariants of algorithms and to create containers for source code generation for SADT diagrams for the other data structures (direct access files and sequential files, strings or trees).

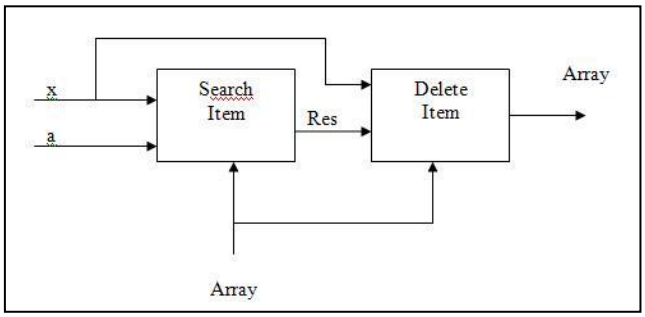


Figure 7. Combining the basic algorithm A3 and A4.

In Fig. 8, the invariants of these algorithms are marked in bold.

f=0;
Res=-1;
 	
i=0;
while(i<strlen(str)
&& !f)
{
r= str[i];
if (r==a)
{
f=1;
Res=i;
}
else
i++;
}
Algorithm A5
	f=0;
	Res=-1;
	fseek(file, 0L, SEEK_SET);
	i=ftell(file);
	while ((fscanf(filein, "%d", &r) !=
	EOF) && !f)
	{
	i=ftell(file);
	if (r==a)
	{
	f=1;
	Res= i;
	}
	else
	i++;
	}
	Algorithm B5

Figure 8. Element search (string and file).

The project results will be used for teaching the bachelors "Software Engineering" in the discipline of "Programming Fundamentals."

Prospects for the development of the project involve the solution of such problems:

- analyze generated programs to determine their efficiency;
- add an interface arc "Control" to describe conditions that are imposed on standard algorithms.

IX. Conclusion

In learning the programming fundamentals of "Software Engineering" students is necessary to develop their competence of structuring problems. When studying standard algorithms and basic data structures it is important to show the general approaches and implementation features. Therefore, this project focuses on the using of SADT-methodology for learning the programming fundamentals. Possibilities of the specification are limited to standard algorithms and basic data structures. Approach of source code generation by SADT specification is offered. Invariants are allocated in basic algorithms. Source code generation algorithm on the basis of templates is formulated. Possibility of creation of new algorithm on the basis algorithms is provided. Data structures used for implementation of algorithm are considered. Internal representation of the SADT specification can be used to analyze the properties of the generated program.

References

[1] D. Ross, “Structured Analysis (SA): A Language for Communicating Ideas”, IEEE Transactions on Software Engineering, vol. SE-3, N. 1, pp. 16-34. 1, Jan. 1977.

[2] D.A. Marca and C.L. McGowan, "IDEF0 and SADT: a modeler's guide,", Auburndale, OpenProcess, Inc., 2006, p.392.

[3] “Recommendations for teaching software engineering and computer science in universities = Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering; Computing Curricula 2001: Computer Science,” V.L. Pavlov, A.A. Terekhov, and A.N. Terekhov, Eds. Moscow: INTUIT.RU "Online University of Information Technologies", 2007, p. 462 [Рекомендации по преподаванию программной инженерии и информатики в университетах, М.: ИНТУИТ.РУ "Интернет-Университет Информационных Технологий", 2007, 462с.].

[4] S. Kelly, J.-P.Tolvanen, "Domain-Specific Modeling: Enabling Full Code Generation," Wiley-IEEE Computer Society Press, 2008, p.448.

[5] J.-P. Tolvanen, “Domain-specific modeling for full code generation,” Journal of Software technology, vol. 12, N. 4, Jan. 2010.

[6] B. Jager and M. Rosenau, “Method for generating source code in a procedural, re-entrant-compatible programming language using a spreadsheet representation,”. US patent application 11/057,430, issue date 9/6/2011, patent number 8015481.

[7] N. M. Jakubiak and M. Kucharski, “System and method for generating source code-based test cases,” application number 11/558241, publication date 08/16/2011.

[8] J. M. Festa,”Systems and methods for generating source code for workflow platform,” patent application number 20100281462, publication date 11/04/2010.

[9] M. Fowler, “Code Generation for Dummies,” Methods & Tools, Spring 2009, vol. 17, N. 1, pp 65-82.

[10] K. Vogel, “A source code generator for C: a language-independent means of building programs that are consistent, elegant, and fast,” Journal Dr. Dobb's, vol. 16, pp. 28 -35, Aug. 1991.

[11] H.S. Delugach, "A Multiple-Viewed Approach to Software Requirements," Ph.D. Dissertation, Department of Computer Science, University of Virginia, Charlottesville, VA, May, 1991.

[12] O. Djebbi, “Eliciting Requirements Variability for Embedded Real-Time System Family,” in Proceedings of the First International Workshop on Situational Requirements Engineering Processes: Methods, Techniques, and Tools to Support Situation-Specific Requirements Engineering Processes (SREP'05), Paris, France, August 29-30, 2005, In Conjunction with the Thirteenth IEEE Requirements Engineering Conference (RE'05), J. Ralyté, P. J. Egerfalk, and N. Kraiem, Eds. Ireland: University of Limerick, 2005, pp. 192-199.

[13] M. Summerfield, "Advanced Qt Programming: Creating Great Software with C++ and Qt 4," Prentice Hall, 2010, p.550.

Technology for creating 3D realtime applications in Android OS

Polotnyanschikov I.S.
mjollneer@gmail.com

Scientific advisor: Zalogova L.A.
zalogova.la@gmail.com

Software and Computing Systems Mathematical Support Perm State University
Perm, Russia

Abstract— **This article discusses the development of technology for creating 3D realtime applications for OS Android. Described tools selection, data domain analysis and realization. Also described process of creating new light model for luminous segment. The results of the work are illustrated with screenshots of real application.**

Keywords-Androd; 3D graphics; Opengl ES; OOP; shaders; lighting model

I. INTRODUCTION

Creation of 3D applications for Android OS became especially actual after devices using this platform spread over the world, outstripping its rival Apple iOS.

Devices with Android OS belong to differing classes of performance and appointment – from a MP3 players and a watches to tablet computers. Users of this devices often require high-quality 3D visualization in real time.

The purpose of the work is to develop technology of creation of interactive applications for the mobile devices using 3D graphics. Achievement of this purpose require to solve the following problems:

- investigate features of development 3D applications in Android OS,
- prove use of an object-oriented paradigm,
- design and implement hierarchy of classes for creation of 3D appendices.

II. CURRENT STATE OF 3D GRAPHICS IN ANDROID

Nowadays best engines of 3D graphics in Android presented by following two categories:

A. Commercial engines

Best commercial engines are Corona SDK and UNITY 3D [9, 10]. No doubt, they offer very powerful cross platform solutions. Library of different multimedia resources, music subsystem, physics emulation, strong animation and most modern visual effects. Customers also get professional support and many tools for development 3D applications. And the cost from $200 to $1500 annually.

B. Free engines

Best free engines are AndEngine and LibGDX [11, 12]. They both corresponds big difficult systems with many functions. Both of them contains many subsystems, like sound, file i/o, animation, physics. LibGDX is crossplatform, AndEngine is 2D-only. Support here is forums and manuals.

Note now, that cross platform source code is noticeably slower than native, because of different wrapping technologies. Also high complexity made mentioned engines difficult to understand and use.

So, taking into account all of the above, seems urgent to develop easy-to-use technology for creating android-only hardware accelerated real-time 3D graphic.

III. INTERFACE CHOICE

There are different approaches to render graphics in Android [7]. OpenGL ES library was chosen as the hardware-software interface. It is recommended by developers of Android for creation of high-efficiency applications [3]. Besides, OpenGL ES allows to reach the most qualitative result.

At present the majority of mobile devices work under control of Android 2.3 or more senior version. In these devices OpenGL ES 1.0 and 2.0 is supported at the same time. So there is a question of a choice.. Each subsequent OpenGL version for the personal computer comprises all functionality of the previous versions. At the same time senior and younger OpenGL ES versions contain essentially different functionality. Therefore, they solve identical problems in qualitatively different ways. For example, the programmer should create a part of functionality of ES 2.0 by means of special programs – shaders. It is impossible to recognize the senior version unequivocally better than the younger. It is necessary to choose the version allowing in the best way to achieve the object of work. This choice will directly affect structure of the developed technology.

IV. OPENGL ES VERSION CHOICE

During comparison of OpenGL ES versions distinctions in their syntax [1,2] and functionality were analysed. Results presented in tab. 1.

TABLE I. RESULT OF COMPARING OF OPENGL ES VERSIONS

Criterion	OpenGL ES 1.0	OpenGL ES 2.0
Code amount	Less	More
Program structure	More easy	More difficult
Result	Worse	Better
Number of supported effects	Less	More
Perfomance	Less	More
Rendering setup	By means of parameters of rasterization, texturing, lighting, etc.	By means of shaders

OpenGL ES 1.0 is suitable when speed of development is more important than image quality. But in this work performance and quality of result interests us first of all. So version 2.0 corresponds us in the best way. In ES 2.0 growth of number of geometrical objects cause the size of the program considerably increases.

Note, that there are different paradigms available for anfroid-developers. For example procedural paradigm and even workable bindings for OpenGL and LISP (functional paradigm) [5,6]. But, for effective management of large amount of a code the object-oriented paradigm was selected.

V. CLASS HIERARCHY DEVELOPMENT

In the course of creation class hierarchy it is necessary to consider features of data domain and the instrument of implementation. The data domain (a 3D graphics) is described in such terms as the camera, geometrical object, a material, a light source. The developed hierarchy includes the classes corresponding to terms specified above. However, their fields, methods and relations substantially depend on features of OpenGL ES 2.0.

For detection of features of ES 2.0 the test program consisting of several geometrical objects, shader objects and attributes of vertexes was written. Specific organization of this program allowed to select repeating parts of a code and data with similar behavior. This features became fields and methods of classes.

The developed object model were named "Lit Engine". In paragraphs A-D classes of LitEngine grouped by implication and described in outline.

A. Creation of geometrical objects

Containers of data – the abstract class LitDataContainer and its successors. They intended to storage attributes of any dimensionality and assignment.

Factory of data – the abstract class LitDataFactory and its successors. They intended to filling data containers with the attributes describing one geometrical object.

A transformation matrix – the class ModelViewMatrix. It is intended to storing and processing matrix in terms of transformation of coordinates.

The universal object – the class Universal3DObject. It is intended for storage all information describing one geometrical object.

B. Light setup

Material – the abstract class LitAppearance and its descendants. They intended to storing settings for the specific shader program.

The effect manager – LitSpecialEffect and its descendants. Effect manger can tune one shader program for one universal 3D object using one specified material."Material".

C. Camera setup

Projection matrix - the class ProjectionMatrix. It is intended to storing and processing matrix in terms of projection of coordinates.

The camera – class LitCamera. It is intended to operate a projection matrix in terms of setup of the camera.

D. Rendering

Shader program – class glslProgramm. It is intended to encapsulate all operation with one shader: loading, compilation, linking, setup and activation.

VI. DEVELOPMENT OF NOT POINTWISE LIGHT SOURSE MODEL

Let’s take a close look on process of development new light model.

A. Formulation of the problem

Lighting calculation by means of shaders traditionally use models of pointwise light sources. For example Ward's models, Lambert, Gooch, Blinn and Phong [13]. These models appeared preferentially to simulate more and more difficult materials.

However, obviously, it is not enough in those situations when it is impossible to neglect the form or the sizes of a source – a lamp shade occupies essential volume and creates dim shadows, the lamp of day lighting shall illuminate like shining cylinder and be mirrored like a bright straight line. The screen of the computer shall be processed as rectangle occupying a certain fixed space and creating an adequate flare on smooth surfaces.

Let's call such objects not pointwise, i.e. consisting of more than one point.

This task is certainly solved by means of global illumination models in which emit and reflect light can any polygon. Such decision provides the high-quality image, generated for the long time every frame. For real-time applications similar physically accurate solution was not found by the author.

Within this work the problem of simulation of not pointwise light sources was posed and solved.

B. Solution

First of all let's describe, how lighting from a luminous segment empirically shall be created.

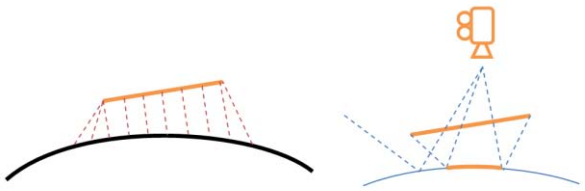


Figure 1. Desirable light for luminous segment

On fig. 1 we see how diffuse (left) and specular (right) light should spread in space near luminous segment.

Note that diffuse lighting is the brighter where the perpendiculars lowered from a segment to a surface.

Author made the assumption that in case of calculation diffuse lighting of each separate fragment it is possible to replace a luminous segment without loss with one pointwise light source. This pointwise source at the same time should belong to a segment and be placed as close to a lighted fragment as possible. Searching of such points on a segment for different surface fragments is schematically figured by orange dotted lines.

Let's note that the mirror flare is brightest on fragments which reflect a vector of a look *v* as precisely into a segment as it is possible (blue dotted lines).

We will make the assumption, based on reasons of common sense, that in case of calculation specular lighting for each surface fragment it is possible to replace a luminous segment with one pointwise light source without loss. This source shall belong to a straight line passing through a segment. At the same time it must be as close to reflected eye vector *v* as possible.

In other words, summary it is necessary to calculate coordinates of two light sources – the first will give us diffusion component, the second – specular.

It is necessary to calculate these coordinates for each surface fragment for every frame.

Specific values of diffusion and specular components can be calculated by means of any pointwise illumination model, for example Blinn.

Below illustrated (fig. 2) a model of diffusion light for segment and one surface fragment.

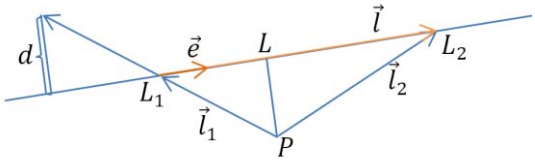


Figure 2. Geometrical model of luminous segment

P – point for which intensity computation is made (surface fragment)

L_1, L_2 – coordinates of the ends of a luminous segment

L – required pointwise diffuse light source

d – distance to a straight line passing through a segment

Computation of required coordinates without intermediate formulas:

$$\vec{e} = \frac{\vec{l}}{|\vec{l}|} \quad (1)$$

$$L = L_1 + \vec{e} \cdot |L_1 L| \quad (2)$$

The received formula is correct, only if the point *L* belongs to a segment. If point *L* is outside of segment, it is necessary to replace it with one of the segment ends.

For computation of coordinates of the second light source it is necessary to find a point on segment close to reflected *v* vector.

$$L = L_1 + m \cdot \vec{l} \quad (3)$$

Equation for coefficient *m* will not be shown in this article. If *m* is not between [0,1], it must be replaced with L_1 or L_2.

Described model was realized via GLES shaders and included in LitEngine.

VII. Example of application of LitEngine

Fig. 3 and 4 is the screenshots of demo application running on mobile device. By means of LitEngine the scene consisting of a set of cubes and illuminated with one luminous segment is rendered in real-time (about 20fps and 600 polygons on Adreno 205 GPU). It is important that luminous segment create adequate diffuse and specular lighting.

Figure 3. Example of use of LitEngine

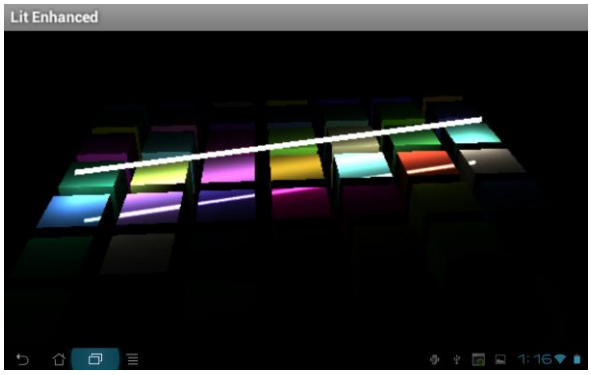


Figure 4. Example of use of LitEngine

VIII. CONCLUSION

Possibility of a reuse of the developed hierarchy was proved in practice by a means of several demonstration applications with a different set of geometrical and shader objects.

Extensibility of the developed hierarchy is proved by one of probable scenarios of extension. For example support of new geometry – the torus was added. Torus was completely incapsulated in one new class inherited from the abstract factory of data. For use of new geometry in the user application its requiring just to rewrite one line of code.

Thus, the developed hierarchy can be reused in case to solve tasks from different data domain (game, simulation modeling, advertizing, editors, GIS, augmented reality, etc.). Further it is necessary to expand model with additional special effects, use JNI [8], include support for external models and create more demo applications.

REFERENCES

[1] Leech J. OpenGL ES Common Profile Difference Annotated Specification 2.0.25. URL: http://www.khronos.org/registry/gles/specs/2.0/es_cm_spec_2.0.25.pdf

[2] Munshi A. OpenGL ES 2.0 Programming Guide. Boston: Addison-Wesley, 2009. 457 c.

[3] Android DevGuide. Graphics URL: http://developer.android.com/guide/topics/graphics/index.html

[4] Imagination Technologies Ltd. Migration from OpenGL ES

[5] Spare time projects: OpenGL and Lisp URL: http://www.mindstab.net/spare-time-projects-opengl-lisp/

[6] Lisp and Android SDK URL: http://stackoverflow.com/questions/5683543/lisp-and-android-ndk

[7] Android Developer. Graphics URL: http://developer.android.com/guide/topics/graphics/index.html

[8] Java Native Interface URL: http://docs.oracle.com/javase/7/docs/technotes/guides/jni/index.html

[9] UNITY URL: http://unity3d.com

[10] Corona URL: http://www.anscamobile.com/corona/index.html

[11] AndEngine - http://www.andengine.org/

[12] LibGDX - http://libgdx.badlogicgames.com/documentation.php

[13] Light models URL: http://steps3d.narod.ru/tutorials/lighting-tutorial.html

Sisal: parallel language development

Idrisov Renat
Program construction and optimization laboratory
IIS Sib RAS
Novosibirsk, Russia

In this paper we present unfinished research on Sisal language currently held at our institute. It wasn't our development initially but now we are trying to create better science computational task solving with it. I also describe why this research is valuable and its current state.

Language description, functional languages, dataflow languages, parallelism

I. INTRODUCTION

Parallel computations are more than actual today, the architecture of the popular computing systems is changed almost every year, a developer needs some universal method of describing a parallel algorithm independent to the system. The idea is to make algorithm description closer to the task description and not to implement the exact algorithm but to formulate the problem as far as it possible. Of course, this idea can be found at early A. P. Ershov articles and is not new in general, but it become more relevant today. Functional programming can give the developer an ability to write programs like problem statements and it is better for exploration of parallelism. In this article we briefly describe some of Sisal programming language constructions and its benefits for parallel algorithm forming.

II. ADVANTAGES

A. *Single assignment*

Sisal [1] [2] differs from other functional languages and we think that this difference make Sisal more adapted for computational tasks. First of all, it has some usual functional language benefits like single assignment[3]. This approach requires every variable to be defined only once. Someone would say that it is not an advantage because every imperative program can be converted to SSA-form, and of course at low-level programming it has no difference but imagine some function and the global variable in the language where every variable need to be declared (we use C for example):

```
int g=0;

void foo(void) { g=1; }
```

You need to re-declare the global variable when it is modified, but you can't make it inside the function. Inside the compiler this program will be converted quite easy but to write initially singe assignment programs is not the same. You can declare another global variable without setting any value but it can bring more questions to the rest of the code, we can use more complex example to withdraw this but we wouldn't. The idea is that single assignment is something similar to structural programming where "goto" operator is prohibited.

B. *Streams and arrays*

Sisal also uses arrays and loops which is not common for a functional language, but it is good for computation: you don't have to worry about the recognition of the tail recursion or the number of iterations or matrix description which is simpler with arrays.

You can operate with n-th element of the array in a natural way like in Fortran:

```
for i in 1, N repeat
        R := A[i] * B[k]
    returns array of R
```

C. *Verbose syntax*

And the last benefit is more verbose syntax. It makes program source more readable and as the result – long time development by different people becomes easier. Many functional languages suffers from the lack of the words in the program source, it makes the text hard to understand. The example below is the famous Haskell1 quicksort:

```
qsort []     = []
qsort (x:xs) =
qsort [y | y <- xs, y < x] ++ [x] ++ qsort [y | y <- xs, y >= x]
```

This kind of code is hard to maintain. The same algorithm implemented in Sisal listed below:

```
function qsort (Data : array[real] returns array[real] )
if array_size( Data ) > 2 then
let
L, Middle, R := for E in Data
 returns array of E when E < Data[ 1 ]
array of E when E = Data[ 1 ]
array of E when E > Data[ 1 ]
 end for
in
```

[1] More information on Haskell can be found at http://www.haskell.org

```
qsort( L ) || Middle || qsort( R )
end let
else
Data
end if
end function
```

III. LOOPS AND REDUCTIONS

In functional programming every statement is a function returning the value, the loops are the same. Reduction is used to determine the returning value of the loop. Keyword "returns" at the end of the loop is followed by the name of the reduction and its parameters. For example, if we need to summarize the elements in the array or the stream we use following construction of the loop:

```
function sum(A: array[real] returns real)
for r in A
returns sum of r
end for
end function
```

Of course, loop construction can be used without any function declaration. Sisal is pure functional, it has no side effects and any loop contains the reduction call, also user can implement his own reductions.

The reductions are good because its implementation can depend on target system. When the program is executed in single-threaded environment it can be performed sequentially, but when executed on multiple threads it can be performed in parallel. Similar idea can be found in modern library "Threading Building Blocks" by Intel[2]. This library allows usage of reduction mechanism in C++, but user can also use ordinary loops as well. In Sisal programs reductions can't be avoided.

In Sisal we have three kinds of loops: Post-conditional, pre-conditional and "for all" (operation is applied to the set). Reductions can be folding or generating (some aggregation function or an array generator). Conditional loops are sequential in general but reduction allows them to be pipelined easier “Fig. 1”.

At this figure loops are divided into parts: Initialization, loop body, loop test, loop reduction (ret) and range generator, we think that the part names can briefly describe them, but if you need more information – please check Sisal language description [1] [2].

Using reductions matrix multiplication can be implemented meaningfully:

[2] More information can be found at http://threadingbuildingblocks.org/

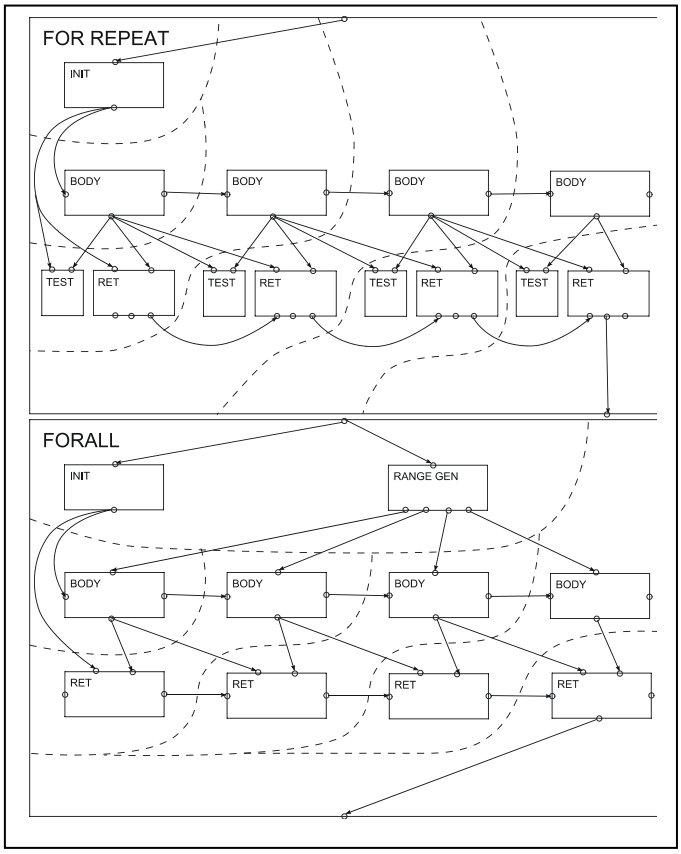


Figure 1. “for all” and post-conditional (for repeat) pipelined structure

```
function multiply( A,B: array[array[real]];  M,N,L : integer
returns array[array[real]] )
  for i in 1, M cross j in 1, L
  returns array of
    for k in 1, N repeat
      R := A[i,k] * B[k,j]
    returns sum of R
    end for
  end for
end function
```

Reduction can be always used in sequential style:

```
function multiply(A:array[array[real]]; B:array[array[real]];
N:integer
            returns array[array[real]])
  for i in 1, N cross j in 1, N
  returns array of
    for initial
      s := 0.0;
      k := 1
```

```
        while (k <= N)
        repeat
          s := old s + A[i, old k] * B[old k, j];
          k := old k + 1
        returns value of s
        end for
      end for
    end function
```

But imperative languages doesn't have any reduction mechanism at all

IV. ERROR HANDLING

Try-catch mechanism is more popular for error handling today but this approach has conflicts with parallel program execution. When the exception occurs all the execution streams must be stopped, pipeline flushed and so on. Also it is harder to keep program determinism in the case of the parallel execution and exception occurs. Check the following JAVA example:

```
try {
      for (int i=0;i<N;i++) {
            a[i]=a[i]/((i+1)%K);
      }
} catch (Exception e) {
      // display partial results stored in "a"
}
```

In this example loop iterations are independent and can be executed in parallel. Sequential execution will always give the same result (for the fixed values of N and K); the result will not depend on the executor properties as far as it remains to be sequential. While there is no dependence between the iterations, programming language semantics remains to be sequential and parallelism exploration can break this semantics or demand additional corrections to keep it. Interpreter or parallelizing compiler needs additional mechanism to differ between the data before and after the exception.

In Sisal language we have “always finished computations” semantics, which means that execution stream will not stop on any error and return resulting value even if the error occurs (“Fig. 2”).

V. RELATED WORKS

New parallel language development is not popular today; more popular is existing language extension (sometimes it is positioned as a separate language); such approach keeps sequential semantics problems, but considered as the fastest both for the developer and for the final application execution. In this section we will not observe such extensions as related.

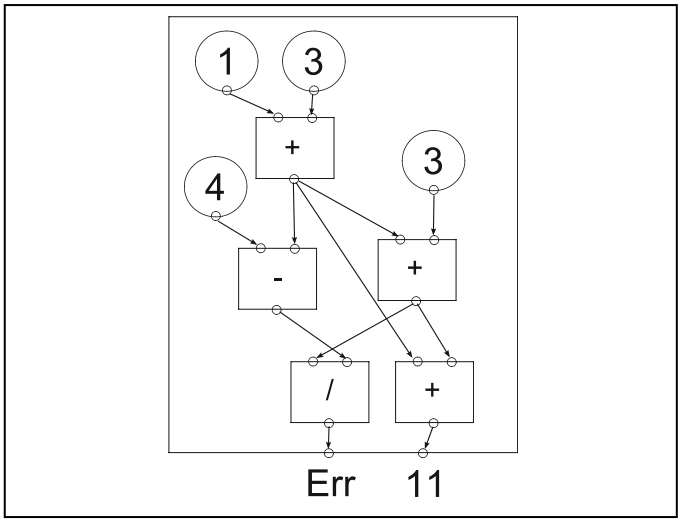


Figure 2. Error value propagation in “always finished computations” semantics

A. *Pifagor*

This language is currently developed at Siberian Federal Institute [4]. The language is optimized to dataflow graph description; syntax is not easy to understand because it differs from common imperative and functional languages. For example, it has no infix operations, no loops. The following Pifagor function performs vector multiplication by scalar:

```
VecScalMult << funcdef Param
// Argument format: ((x1, x2, : xn), y),
// where ((x1, x2, : xn) is a vector, y – scalar
{
((Param:1,(Param:2,Param:1:|):dup):#:[]:*) >>return
}
```

It is hard to compare Pifagor syntax and constructions with Sisal because they are completely different. Sisal has loops and arrays; we suppose it is better for science computational tasks. According to the articles of the Pifagor developers it is aimed on the list processing and the conception of unlimited parallelism scheduled as limited at runtime.

This project has compiler and interpreter used for scientific proposes: development of the new scheduling algorithms and parallel programming education.

B. *F# from Microsoft*

We can’t say that F# is the project in a same direction with Sisal, but Microsoft’s developments in a functional paradigm can’t be avoidable. As the complexity of the systems was increased the complexity of compiler grows and some features of the functional languages formerly considered as ineffective started to implement in imperative languages.

At one hand: F# is functional ML-family language; functional paradigm suits better for parallel computations. At the other: it has an ability to create any mutable indexes, non-functional calls or dependencies, external .NET objects and operations. It can’t be considered as single assignment or parallel; it is hybrid, you can write implicitly parallel and sequential programs both. Multithreaded programming on F# is quite similar to C# or C programming.

Not in case of the only F# but for the all functional languages developers are trying to make language programming available for wide range of people but it makes language less pure and less functional. State modification operators such as input and output give the developer familiar ability to process the data but makes the semantic sequential or non-deterministic.

VI. Our current state

At our institute we develop both language standard and compiler. First version of the language was derived from Sisal 1.2 initially developed at Lawrence Livermore National Laboratory [5]; current version of the language is 3.2 and we are trying to improve it by solving some science computation problems.

Sisal language compiler is used mostly by its developers for scientific proposes: developing new optimization and analysis algorithms, checking and improving language standard.

The main aim for today is to make language available for people who solving computational problems and students. We are developing JavaScript Sisal interpreter to achieve these objectives.

My personal contribution is algorithm implementation for language standard improvement investigations; backend optimization algorithms and parallelization at Sisal compiler. Now I develop JavaScript Sisal interpreter.

VII. Conclusion

The main idea of this paper is to explain some algorithmic solutions and language properties valuable for parallel execution. Development of the new languages become more and more easy, user can create his own science field specific language. And if the programs became task definitions and not algorithm descriptions – it will be not necessary to rewrite it when the execution environment or computation system will change. You have to rewrite only the compiler; it is easier than re-solve all the tasks again. Even if the language will be specific it will give advantages while migrating to another executor. Microsoft pays additional attention to functional programming and provides tools for functional language building, it is positive trend3.

Sisal language was initially developed for parallel programming and writing Sisal programs is not the same as writing C programs. In this paper we briefly described semantic difference. If you are interested in parallel programming and unfamiliar with functional or logic paradigm you should definitely explore it.

References

[1] V. Kasyanov, A. Stasenko, “Sisal 3.2 language structures decomposition”, Lecture Notes in Electrical Engineering. — Berlin: Springer-Verlag, 2009. — Vol. 28. — P. 582–594.

[2] J. McGraw, S. Skedzielewski, S. Allan, D. Grit and R. Oldehoeft “Sisal: Streams and iterations in a single assignment language, Language Reference Manual Version 1.1”, Lawrence Livermore Nat. Lab. Manual M-146. — Livermore, CA 1983.

[3] R. Cytron, J. Ferrante, B. Rosen, M. Wegman and K. Zadeck, “Efficiently computing static single assignment form,” Proc. POPL-1989, pp. 25-35, ACM.

[4] A. Legalov “Functional language for creation of architecture-independent parallel programs” Легалов А. И. Функциональный язык для создания архитектурно-независимых параллельных программ Вычислительные технологии : журнал. — 2005. — Т. 10. — № 1. — С. 71-89

[5] McGraw, J. R. et. al. “Sisal: Streams and iterations in a single assignment language, Language Reference Manual Version 1.2” Lawrence Livermore Nat. Lab. Manual M-146 (Rev.1). — Livermore, CA 1985.

[3] This tools are provided with F# can be found at http://research.microsoft.com/en-us/um/cambridge/projects/fsharp/

MetaLanguage: a Tool for Creating Visual Domain-Specific Modeling Languages

Alexander O. Sukhov
Department of Software and Computing Systems Mathematical Support
Perm State University
Perm, Russian Federation
E-mail: Sukhov.psu@gmail.com

Scientific Advisor:
Lyudmila N. Lyadova
Department of Business Informatics
National Research University Higher School of Economics
Perm, Russian Federation
E-mail: LNLyadova@gmail.com

Abstract. **The technologies based on applying a metamodeling and domain-specific languages are widely used at information systems developing. There are many different tools for creating graphical domain-specific language editors with a possibility of determining user's graphical notations. However they possess disadvantages. The MetaLanguage system is designed to eliminate some of these shortcomings. MetaLanguage is a language workbench which provides creation of visual dynamic adaptable domain-specific modeling languages used in the development of information systems. In paper the approach to development of MetaLanguage DSM-platform is considered. Basic metalanguage constructions of this system are described. The mathematical multilevel domain model with usage of pseudo-metagraphs is constructed. Definitions of the graph and metagraph are given. The algorithm of vertical models transformations is described. The architecture and implementation of the development environment of MetaLanguage toolkit is presented.**

Domain-specific language; DSM-platform; MetaLanguage; metamodel; visual modeling languages; graph grammars

I. INTRODUCTION

One of the key requirements for information systems is the possibility of flexible customization to ever-changing needs of business processes and users. Domain modeling is an essential stage in the development of any information system. One of approaches for maximum adaptability – using models not only at the system development stage, but also at system functioning.

Model is an abstract description of system characteristics which are important from the viewpoint of modeling purposes. Model is described in some formal language. To each task solution can be applied a modeling language which uses concepts and relations from the information system domain. The systems life cycle is based on usage of the several models that are described from the various points of view and with different levels of abstraction. Such approach is caused by that system development process consists of several stages: analysis, design, implementation, testing. For example, at the analysis stage on the software look as on implementation of specific business functionality needed to the customer, herewith principles and details of implementation are not important.

At system creation several levels of models are created: the data that are stored in system database is a state model of the information system domain; their description, which providing a data interpretation or code generation to work with them, is a metamodel; for developing this model special formal language, which allows to work in terms of the appropriate domain, is applied – the meta-metamodel here is used.

In fact, system creation with usage of modern workbenches represents the development of domain-specific languages (DSLs) – information system meta-metamodels. DSLs are simple on applying and are easy to understand for users as they operate with domain terms. Therefore now a large number of DSLs is developed for using in different domains, for example, for business processes modeling [1] and the designing applications for mobile devices [2].

The use of DSLs and language workbenches allows to simplify process of models creation. Experts – specialists in various domains can be involved in the development. Expressiveness of languages and productivity of the systems created on their basis depends on properties of baseline models, a choice of mathematical formalism for describing language properties.

Today, there are many widespread visual DSLs, because the diagrams are more clear and understandable not only for programmers, but also for the domain experts and system users. This approach to use of visual DSLs is called domain-specific modeling (DSM). *DSM-technology* provides modeling in domain terms.

There is no unified general-purpose visual language of software development. In practice now are widely used such languages of visual modeling, as Class Diagrams and ERD – for domains modeling; IDEF, DFD, EPC, BPEL, and BPML – for business process modeling, etc.

This paper is supported by Russian Foundation for Basic Research (Grant 12-07-00763)

Recently, UML claims to be the modeling language standard, however, this language has some significant disadvantages:

- UML diagrams are complicated for understanding not only for experts who take part in system engineering, but in some cases even for professional programmers;
- UML diagrams can't adequately represent domain concepts, since work is being done in terms of "class," "association," "aggregation," etc., rather than in domain terms.

The language used to create other languages is called the *metalanguage*. Process of model creation can be iteratively: having created some language, we can use it as a metalanguage for designing other language which, by-turn, also can be used as a metalanguage, etc.

Despite all DSL advantages they have one big disadvantage – complexity of the designing. If general purpose languages allow creating programs irrespectively to domain, in case of DSLs for each domain, and in some cases for each task it is necessary to create the domain-specific language. If the domain is quite simple and language is uncomplicated, the compiler will create easily. More complex domain and language will require much effort. Another shortcoming of domain-specific language is that it's necessary to create convenient graphical editors to work with it.

The *language workbench* or *DSM-platform* is the instrumental software intended to support development and maintenance of DSLs [3]. Usage at DSLs creation a language workbench considerably simplifies the process of their designing [4].

It is necessary to make following demands to tools that are using for creation of visual DSLs:

- possibility of modeling languages defining for the majority of domains, as for description of business processes, ontologies, object models, and for models of applications for mobile devices creation;
- unified representation and description of models and metamodels, i.e. for models and metamodels definition the same toolkit should be used;
- ability to dynamically change the language description without source code modification and without system restart;
- consistency of domain metamodels and models description, i.e. system should support language and models in a consistent state, and when metamodel changes system must perform all necessary modifications in corresponding models automatically;
- enabling an ability of iteration metamodels definition, i.e. describing a metamodel, the developer should be able to use it as a tool for creation other metamodels;
- possibility of models transforming from one notation to another.

II. Related Works

There are many different DSM-platforms for developing DSLs graphical editors with a possibility of determining user's notations. These tools are MetaEdit+, MS DSL Tools, Eclipse GMF, State Machine Designer, Meta Programming System, REAL-IT, UFO-toolkit, etc. A main idea of DSM-approach is to create toolkits that support optimal variants of visual modeling for specific domain. Let's consider these platforms in more detail.

UFO-toolkit [5], unlike the other systems, supports a simulation modeling of created models. This tool provides a representation of any system as a set of three-element constructions: "Unit – Function – Object" (UFO-element). The "Unit" is a point of intersection of input and output arrows. The "Function" is a transformation process of input into output. The "Object" is a substance that implements this function. The disadvantage of this system is that it does not support a possibility of models usage created in other systems since its notation does not correspond to an open standard.

Technology REAL-IT [6] is based on the use of UML. Information system development is reduced to description of the database and user interface with CASE-package REAL. On the basis of these models the application can be automatically generated. The generation possibility is provided by user interface standardization and lack of nontrivial logic of data processing. Otherwise in the generated code it is necessary to add the code written "by hand."

REAL-IT and UFO-toolkit at information systems creating allow using only the built-in modeling languages. This significantly limits the customization of these systems.

MetaEdit+ is a multiplatform environment that enables users to simultaneously work with several projects each of which can have a few models [7]. At usage this DSM-platform besides a possibility of domain-specific language creation, the developer receives the CASE tool into which this language is integrated. MetaEdit+ allows to use several DSLs at system creation.

The approach based on metamodels interpretation, instead of code generation used in MetaEdit+ allows changing the DSL definition at run-time. The system allows working with languages and metalanguages universally, using the same tools. The disadvantage of MetaEdit+ is that this DSM-platform for export of models uses an own file format (MXT) and this affects the openness of technology.

DSL Tools [8, 9] and Eclipse GMF [10, 11] technologies provide the user with advanced IDE MS Visual Studio and Eclipse, respectively. Because of this there is a possibility of code completion in high-level languages "by hand," but it can lead to inconsistency of diagrams and source code. State Machine Designer [12], in fact, is an add-on DSL Tools, eliminating some of its defect. However, the State Machine Designer allows creating a DSL only using UML Activity Diagrams that considerably limits the range of tasks.

As opposed to other DSM-platforms in the Meta Programming System [13] a method for designing textual

DSLs is supported. It's not so convenient, because the text is not sufficiently expressive.

Technology Eclipse GMF is most powerful of the above. However, its use is impeded by the lack of documentation, complexity, and frequent releases of new versions. In fact, Eclipse GMF is in a stage of intensive development.

Eclipse environment provides the user with tab GMF Dashboard which allows accelerating DSL development process by automatically generating of some language components. On GMF Dashboard tab the sequence of the operations which execution will lead to creation of a plug-in for Eclipse that allows to build diagrams in current domain is represented.

Cases when DSLs becomes part of other applications are common. For example, a specially designed language for describing business processes can be used in document circulation. Therefore one more important characteristic of the DSM-platforms is their alienability of the development environment. DSL Tools, Eclipse GMF, Meta Programming System are strongly associated with the development platforms – MS Visual Studio, Eclipse, IntelliJ-IDEA, respectively, therefore languages created by these workbenches can't be exported to external system.

All of these technologies do not provide the ability to create both visual and textual DSLs. In addition, all DSM-platforms, except for the MetaEdit+, do not allow creating the dynamic adaptability languages.

Existing problems of definition and using domain-specific visual modeling languages and DSM-platform restrictions became a reason to the MetaLanguage system creation, which would integrate the advantages of existing language workbenches and eliminate some disadvantages.

The visual metalanguage of created system should

- allow to build models that are sufficiently detailed and accurately describe the domain, so detailed and accurately how much it is necessary in each case, thus for different detail levels of description it is necessary to use the same constructions;
- have a simple constructions, allowing to work with the metalanguage not only to professional programmers but also ultimate users, such as business analysts;
- provide an opportunity to specify not only language syntax, but also its semantics.

III. CONSTRUCTIONS OF METALANGUAGE SYSTEM

The main shortcoming of metalanguages, which are used for DSLs designing, is their static character: the developer can't change the existing metalanguage constructions. A basis of this problem is that the metalanguage description is embedded in system source code, therefore for metalanguage modification it is necessary either to modify the source code, what to make in most cases impossible, or to offer to put up with language capabilities.

If the metalanguage description will be presented in the form of metadata, there will be possibility to change created language constructions in dynamics, i.e. without modification of system source code.

MetaLanguage system is a tool for creating visual dynamic adaptable domain-specific modeling languages used for development of information system. To describe the metamodels MetaLanguage toolkit uses metalanguage, which basic constructions are the entity, the relation, the constraint.

A. Entity

The *entity* is any construction of modeling language. Entities are characterized by

- name that uniquely identifies the entity within the metamodel;
- amount of entity instances that can be created in the model;
- set of entity attributes;
- set of entity operations;
- set of constraint imposed on the entity;
- flag of uniqueness that determines limits of entity instance name uniqueness.

The amount of entity instances defines how many instances can be created in the model. The amount of instances is set by an integer from the interval $[0, \infty)$. If value of this entity characteristic is equal to zero, then at model designing the entity of this type will not be in list of entities, proposed for creation. If the value of the characteristic is equal to infinity, it is possible to create an arbitrary number of this type entity instances.

Attribute is the named property of the entity (relation), including a description of valid values set.

The attribute has

- name that uniquely identifies it within the entity (relation);
- type that determines a set of possible values for the attribute and the operations that can be done on its values;
- default value which will be chosen as the attribute value, if the last is not specified;
- description which contains some additional information about the attribute.

Entity (relation) can have any number of attributes or not have them at all.

Operation is an abstraction of actions which can be carried out over the entity. In most cases, an applying of the operation leads to the fact that the entity changes the state.

The operation includes:

- name that uniquely identifies the operation within the entity;
- operation parameters;
- default values for parameters which in case of unavailability of basic values will be used when an operation call;
- type of returned value;
- description, containing the additional information about the operation.

Entity can have any number of operations or not have them at all.

Consider the examples of entities. Fig. 1 shows a fragment of metamodel for UML Use Case diagrams. The metamodel contains two entities "Actor" and "Use Case."

The entity "Use Case" has following attributes: "Name," "Description," "Creation_Date." The attribute "Name" has a string type and defines the Use Case name. The attribute "Description" sets the short description of the Use Case. "Creation_Date" – the attribute which contains information on when the "Use Case" has been created. Over the entity "Use Case" the following operations are admissible: "SetName()," "SetDescription()," "SetDate()."

An attribute of "Actor" is a string attribute "Name" which specifies the name of the actor. Permissible operation over the entity "Actor" is the "SetName()"operation.

B. Relation

Visual languages constructions in rare cases exist independently, more often they are in some way related to each other, therefore at metamodel creation importantly not only to define the basic language constructions, but also correctly specify the relations between them.

The *relation* is used for description a physical or conceptual links between entities.

Any relation is characterized by

- name that uniquely identifies the relation in this metamodel;
- type that defines the semantics of the relation;
- set of relation attributes;
- set of constraint imposed on the relation;
- multiplicity which determines how many entity instances can participate in the relation;
- flag of uniqueness that determines limits of relation instance name uniqueness.

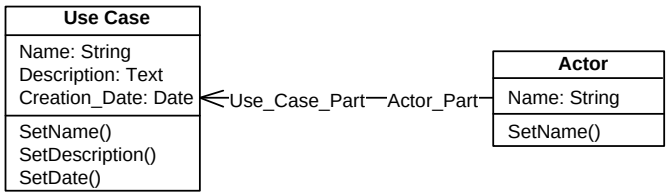


Figure 1. Fragment of metamodel for UML Use Case diagrams

The metamodel can contain the following types of relation: inheritance, association, aggregation. However in models it is possible to create only instances of the association and aggregation relations. Consider each type of relation in more detail.

Inheritance – a relation between the general entity (superclass, parent) and a specific entity (subclass, child).

The child entity inherits all parent attributes, operations and relations. In addition to the parent it can also have their own attributes, operations, relations, therefore child entity can be used everywhere where the parent entity is used, but converse is not true.

Entity can have only one parent and unlimited number of child entities, i.e. multiplicity of this type relation is 1:M.

On Fig. 2 the fragment of metamodel for Entity-Relation Diagrams is presented. The metamodel contains the entities "Abstract," "Attribute," "Entity," "Relation." In order to reduce the diagram entity operations are not represented in figure.

Attributes of the entity "Abstract" are "Name" that identifies an entity instance, and "Description," containing the additional information about the entity.

The entity "Attribute" has following attributes: "Name," "Type" and "Description."

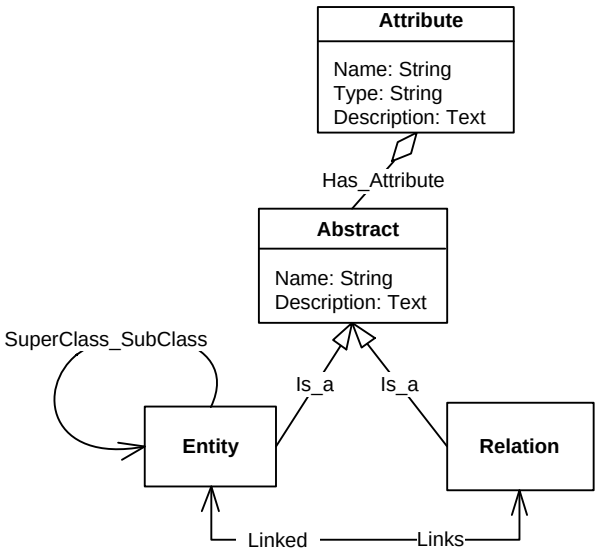


Figure 2. Fragment of metamodel for Entity-Relation Diagrams

The entity “Abstract” is abstract, i.e. it is impossible to create instances of this entity in the model. “Abstract” acts as a parent for entities “Entity” and “Relation” (in the figure it is shown by an arrow with a triangular end). Both child entities inherit all parent attributes, operations, relations; these entities have no own attributes and operations. Entities “Relation” and “Entity” in addition to the inherited relation “Has_Attribute” have their own relation “Linked_Links.” Another association relation “SuperClass_SubClass” belongs to the entity “Entity.”

Association is a structural relationship which specifies that entities of one kind are connected to entities of another.

If two entities are connected by association, then we can navigate from one entity instances to another entity instances. The association relation can be unidirectional and bidirectional. Unidirectional association is used, when it is necessary to specify that the relation instance can be drawn only in the given direction, bidirectional association defines that the relation instance can be drawn in both directions. The case when both ends of association belong to one entity is a valid. It means that some entity instance can be associated with another instance of the same entity.

In addition to the previously described basic characteristics of the relation, there is one more which applies only to the association – a role. Entities related by association plays a role in it. The *role* is a name which uniquely identifies one of the association ends.

The arbitrary number of entity instances can participate in association as with one, and on the other hand, thus, generally a multiplicity of this relation is M:M.

On Fig. 2 two associations are presented. The bidirectional association connects entities “Relation” and “Entity” it means that in ERD-models between these entity instances it is possible to draw equivalent relation. The second unidirectional association binds entity “Entity” with itself, this allows any instance of “Entity” to have parent (another instance of “Entity”) in ERD-models.

Aggregation – a kind of association that models an unequal part-whole relation.

The main difference of aggregation from association is that the last reflects the relation between two equal entities, while in aggregation one of entities is the main and another – dependent. The distinctive features of aggregation is also the fact that this type of relation is always directed, the multiplicity of this relation is 1:M, and the aggregation ends can’t belong to one entity.

At removal of main entity instance all instances of dependent entity participating in this aggregation will be automatically deleted.

In ERD metamodel between entities “Abstract” and “Attribute” the aggregation relation is set (in figure this relation is represented by an arc with a diamond end), therefore in ERD-models instances of entities “Relation” and “Entity” can be connected by aggregation with the instances of entity “Attribute.”

C. Constraints

In practice quite often there are cases when it is necessary to impose any constraints on entities and relations between them.

If rules of diagrams connection set syntax of visual language, constraints define its semantics. Some of constraints are set by metamodel structure, and others are described on some language. An example of the language used to describe constraints is OCL.

All constraints imposed on the metamodel can be divided into two groups: constraints imposed on the entities and constraints imposed on the relations.

Constraints imposed on the entity can be one of the following types:

- constraints imposed on the uniqueness of entity instance name;
- constraints imposed on the amount of entity instances in model;
- constraints imposed on the attribute values of entity instance.

The name of the entity instance can be unique in the metamodel, in the model or not be unique. The uniqueness in the metamodel means that in all models which are created on the basis of a current metamodel the entity instance name should be appeared only once. The constraint of such type it is necessary to set on the “Use Case” entity of metamodel for UML diagrams, if you want to specify that names of all instances of the “Use Case” entity must be unique in all models.

The uniqueness in the model means that the name of entity instance will be unique only within limits of the model of which this entity belongs. The condition of name uniqueness of the “Actor” entity in the Use Case diagram model can be an example of such constraint.

Constraint imposed on the amount of entity instances in model is set by specifying the number of instances at entity creation. So instances of abstract entities at which value of property “amount” is equal to zero, will not participate at model creation. If value of this property is equal to one, then in model it is possible to create only a single instance of this type entity. An example of this type constraint is a condition that limits an amount of created instances of the entity “Actor” by value five, it will build a clear diagram, which is not encumbered by great number of “Use Cases” and “Actors.”

In terms of defining the semantics of visual language the constraints imposed on the attribute values of entity instance are the most important. Such constraints are specified as triples:

Attribute_Name: *Sign*: *Value.*

“Value” can be a constant, attribute value of the entity instance or some function of attribute values of entity instances.

For example, in a metamodel of Use Case diagrams constraint of this type can be imposed on the attribute "Creation_Date" of the "Use Case" entity, because the date can't exceed the current time. Such constraint may look like:

Creation_Date <= Now(),

where function Now() returns current system time.

All *constraints imposed on the relation* may be divided into following groups:

- constraints imposed on the uniqueness of relation instance name;
- constraints imposed on the types of connected entity instances;
- constraints imposed on the relations multiplicity;
- constraints imposed on the attribute values of connected entity instances.

Constraint imposed on the uniqueness of relation instance name are similar to constraint imposed on the uniqueness of entity instance name and can accept one of values: unique in the metamodel, unique in the model, non-unique.

Constraints imposed on the types of connected entity instances are defined by metamodel structure. These constraints set rules for connection of different types of entity instances. For example, the metamodel in Fig. 1 hasn't association the ends of which belong to the same entity, this means that between two instances of the "Use Case" entity or between two instances of the "Actor" entity it is impossible to create an association instance.

Constraints imposed on the relations multiplicity are set at their creation. Thus the relation of inheritance and aggregation supports only 1:M multiplicity, which can be adjusted only for dependent entity multiplicity. The association admits M:M multiplicity with the ability to refine.

If in models of Use Case diagrams it is necessary to specify that the amount of the "Actors" which involved with "Use Case" can't be more than five, then at creation of association between entities "Use Case" and "Actor" it is necessary to set the M:5 multiplicity.

The constraints imposed on the attribute values of connected entity instances carry the greatest semantic weight. Difference of these constraints from the constraints imposed on the attribute values of entity instance is that first type constraints allow setting specific entity instances on which constraints are imposed.

Constraints of this type can be set on values of attribute "Birthday" of connected entities "Person" in constructing the metamodel "Family tree," as the parent's birthday can't exceed of child's birthday.

IV. Mathematical Description of Multilevel Domain Model

Using constructions entity and relation it is possible to build any model, including an invalid in the current domain. There are various formalisms for specifying the syntax of visual languages: automatic models [14], algorithmic nets [15], graph grammars [16], et al.

Most of the existing approaches to definition visual languages syntax consider a concrete syntax, and only in rare cases – abstract syntax. The abstract syntax of visual modeling languages does not need all those details that are presented in a concrete syntax: it is possible to abstract from the choice of icons used to display the language elements, and their geometrical parameters, etc.

To define the formal rules of models creation it is proposed to use graph grammars. Graph grammar is a generalization of Chomsky grammars on graphs. To define a grammar it is required to specify the finite sets of terminal and nonterminal symbols, a finite set of production rules, and select the start symbol in nonterminal symbols set. For representation graph grammars it is necessary to choose such type of graphs which would be provided the opportunity for an iteratively metamodels definition, unified representation and description of domain models and metamodels.

Production rules in graph grammar contain the left- and the right-hand side. If to generalize the classic definition of graph grammars, then as right-hand side of the rule may be not only a labeled graph, but the code in any programming language, and also a fragment of a visual model described in other notation. That is why the graph grammar can be used for generation syntax correct models and for refactoring of existing models, code generation and model transformations from one modeling language to another [17].

As an analysis result of various representations of graph grammars it was determined that the most appropriate formalism for describing the syntax of visual modeling languages in MetaLanguage system are graph grammars, which are constructed on the pseudo-metagraphs [18]. Let's define the domain metamodel and model, applying the selected formalism, and construct the direct and reverse map of metamodel graph on model graph.

A. Metamodel graph

Let $Ent = \{ent_i\}, i \in \mathrm{N}, i < \infty$ (N – set of natural numbers) is a set of metamodel entities that is finite at every fixed point in time, but extends at entity creation and reduces at removing.

Let's designate each entity as a tuple

$ent_i = \{EName_i, EICount_i, EAttr_i, EOpp_i, ERest_i, EUnique_i\}$,

where $EName_i$ is a entity name, $EICount_i$ – amount of entity instances, $EAttr_i = \{eattr_{j_i}\}, j_i \in \mathrm{N}, j_i < \infty$ – entity attributes, $EOpp_i = \{eopp_{j_i}\}, j_i \in \mathrm{N}, j_i < \infty$ – entity operations, $ERest_i = \{erest_{j_i}\}, j_i \in \mathrm{N}, j_i < \infty$ – set of constraint imposed on the entity, $EUnique_i$ – flag of uniqueness.

Sets $EAttr_i$, $EOpp_i$, $ERest_i$ are finite at every fixed point in time.

Let's divide all characteristics of i-th entity on two groups EG_i^1 and EG_i^2. The first group consists of those characteristics,

which will be represented by separate nodes in graph model: sets of attributes, operations, and constraints imposed on the entity, i.e.

$$EG_i^1 = \{EAttr_i, EOpp_i, ERest_i\}.$$

Characteristics of second group $EG_i^2 = \{EName_i, EICount_i, EUnique_i\}$ (entity name, amount of entity instances, flag of uniqueness) will be attributed to node of the corresponding entity directly.

$Rel = \{rel_i\}, i \in \mathrm{N}, i < \infty$ denotes a set of metamodel relations that is finite at every fixed point in time, but extends at relation creation and reduces at removing.

Let relation is a tuple

$$rel_i = \{RName_i, RType_i, RAttr_i, RMult_i, RRest_i, RUnique_i\},$$

where $RName_i$ is a relation name, $RType_i$ – relation type, $RAttr_i = \{rattr_{j_i}\}, j_i \in \mathrm{N}, j_i < \infty$ – relation attributes, $RMult_i$ – multiplicity, $RRest_i = \{rrest_{j_i}\}, j_i \in \mathrm{N}, j_i < \infty$ – relation constraints, $RUnique_i$ – flag of uniqueness.

Sets $RAttr_i$, $RRest_i$ are finite at every fixed point in time.

Characteristics of i-th relation will be divided into two groups RG_i^1 and RG_i^2. The first group comprises a set of relation attributes and constraints imposed on the relation. The second group includes the following characteristics: "name," "type," "multiplicity," "flag of uniqueness," i.e.

$$RG_i^1 = \{RAttr_i, RRest_i\},$$

$$RG_i^2 = \{RName_i, RType_i, RMult_i, RUnique_i\}.$$

Consider directed pseudo-metagraph $GMM = (V, E)$. Let a set of metamodel graph nodes is a union of seven disjoint subsets:

$$V = Ent \bigcup_{i=1}^{|Ent|} EAttr_i \bigcup_{i=1}^{|Ent|} EOpp_i \bigcup_{i=1}^{|Ent|} ERest_i \bigcup \bigcup Rel \bigcup_{i=1}^{|Rel|} RAttr_i \bigcup_{i=1}^{|Rel|} RRest_i . \quad (1)$$

The set of pseudo-metagraph arcs E divide into six disjoint subsets:

- $EEA = \{eea_i\}, \overline{i = 1, |Ent|}$ – a set of arcs connecting each metamodel entity with set of attributes belonging to it;
- $EEO = \{eeo_i\}, \overline{i = 1, |Ent|}$ – a set of arcs connecting each metamodel entity with set of operations over it;
- $EER = \{eer_i\}, \overline{i = 1, |Ent|}$ – a set of arcs connecting each metamodel entity with set of constraints imposed on it;
- $ERA = \{era_i\}, \overline{i = 1, |Rel|}$ – a set of arcs connecting each metamodel relation with set of its attributes;
- $ERR = \{err_i\}, \overline{i = 1, |Rel|}$ – a set of arcs connecting each metamodel relation with set of constraints imposed on it;
- $EERR = \{eerr_i\}, i \in \mathrm{N}, i < \infty$ – a set of arcs conforming to links between entities and relations that is finite at every fixed point in time, but extends at entity (relation) creation and reduces at removing.

Thus, we see that

$$E = EEA \bigcup EEO \bigcup EER \bigcup ERA \bigcup ERR \bigcup EERR . \quad (2)$$

The *metamodel graph* is a directed pseudo-metagraph $GMM = (V, E)$, for which (1) and (2), where V is a nonempty set of graph nodes, E is a set of graph arcs.

Let's consider an example. We will construct a metamodel graph for the entity "Use Case" of UML Use Case diagrams. Metamodel of this diagram type is shown in Fig 1. Attributes of the entity "Use Case" are "Name," "Description," "Creation_Date." Operations that can be performed on entity – "SetName()," "SetDescription()," "SetDate()," i.e. for given entity

$$EAttr_i = \{\text{"}Name\text{,"} \text{"}Description\text{,"} \text{"}Creation\_Date\text{"}\},$$

$$EOpp_i = \{\text{"}SetName\text{()," "}SetDescription\text{()," "}SetDate\text{()"}\},$$

$$ERest_i = \varnothing .$$

The metamodel graph corresponding to a fragment of the "Use Case" entity shown in Fig. 3.

As can be seen from figure

$$EEA = \{eea_{i_1}\},\ EEO = \{eeo_{i_1}\},\ EER = \varnothing,\ EERR = \varnothing .$$

B. Model Graph

The *model* is actually an "instance" of metamodel in which:

- the attributes of entity – a concrete values;
- there are no operations over entity instances and constraints imposed on the entity and relation instances;
- inheritance relation instances can't be created.

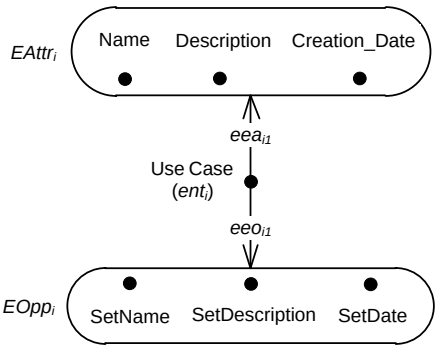


Figure 3. Fragment of metamodel graph for "Use Case" entity

Let's designate a set of all models which have been created based on the current metamodel through $M = \{m_k\}, k \in \mathrm{N}, k < \infty$ that is finite at every fixed point in time, but extends at model creation and reduces at removing.

Let's introduce following notation:

- $EntI_i$ – set of instances of *i*-th entity;
- $EAttrI_{j_i}$ – set of attribute values for *j*-th instance of *i*-th entity;
- $RelI_k$ – set of instances of *k*-th relation;
- $RAttrI_{k_l}$ – set of attribute values for *k*-th instance of *l*-th relation.

Sets $EntI_i$, $EAttrI_{j_i}$, $RelI_k$, $RAttrI_{k_l}$ are finite at every fixed point in time, but extend at entity (relation) instance creation and reduce at removing.

Examine the directed pseudo-metagraph $GM = (VI, EI)$. Let a set of model graph nodes is a union

$$VI = \bigcup_{i=1}^{|Ent|} \left(EntI_i \bigcup_{j=1}^{|EAttr_i|} EAttrI_{j_i} \right) \bigcup_{k=1}^{|Rel|} \left(RelI_k \bigcup_{l=1}^{|RAttr_k|} RAttrI_{l_k} \right). \quad (3)$$

Consider the following example. Let's create a model graph for instance of "Use Case" entity (Fig. 4).

From a figure it is apparently that $EAttrI_i$ = {"*Pass_exam*," "*Use Case describes passing an exam process*," "21/06/09"}.

The set EI divides into three disjoint subsets:

- $EEAI = \{eeaI_i\}, \overline{i = 1, |EntI|}$ – a set of arcs connecting each entity instance with set of attributes belonging to it;
- $ERAI = \{eraI_i\}, \overline{i = 1, |RelI|}$ – a set of arcs connecting each relation instance with set of attributes belonging to it;
- $EERRI = \{eerrI_i\}, i \in \mathrm{N}, i < \infty$ – a set of arcs corresponding to the links between entity instances and relation instances that is finite at every fixed point in time, but extends at entity (relation) instance creation and reduces at removing.

Thus, we see that

$$EI = EEAI \bigcup ERAI \bigcup EERRI. \quad (4)$$

You can see from the Fig. 4 that for represented "Use Case" entity instance $EEAI = \{eeaI_{i_1}\}$, $EERRI = \varnothing$.

The *model graph* is a directed pseudo-metagraph $GM = (VI, EI)$, for which (3) and (4), where VI is a nonempty set of graph nodes, EI – set of graph arcs.

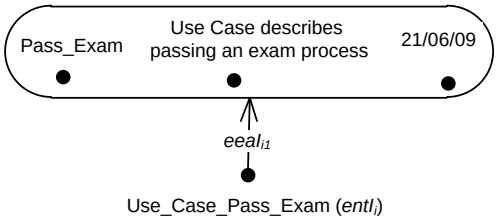


Figure 4. Model graph corresponding to "Use Case" entity instance

C. Operation of Model Graph Creation

Let's construct map of the metamodel graph on the model graph, it corresponds to an operation of a model graph creation. Such map allow to support models in an actual state, as metamodel modification leads to a change of all models created based on it.

Let's introduce following notation:

- $EntI = \bigcup_{i=1}^{|Ent|} EntI_i$ – a set of model graph nodes conforming to all entity instances;
- $RelI = \bigcup_{i=1}^{|Rel|} RelI_i$ – a set of model graph nodes corresponding to all relation instances;
- $EAttrI = \bigcup_{i=1}^{|Ent|} \bigcup_{j_i=1}^{|EntI|} EAttrI_{j_i}$ – a set of model graph nodes conforming to attribute values of all entity instances;
- $RAttrI = \bigcup_{k=1}^{|Rel|} \bigcup_{l_k=1}^{|RelI|} RAttrI_{l_k}$ – a set of model graph nodes corresponding to attribute values of all relation instances.

Sets $EntI$, $RelI$, $EAttrI$, $RAttrI$ are finite at every fixed point in time, but extend at entity (relation) instance creation and reduce at removing.

Let's construct a map that for each metamodel graph entity-node defines a set of model graph nodes conforming to instances of this entity, i.e.

- $(\exists\, ent_i \in Ent)(\exists\, entI_{j_i} \in EntI) : fe(ent_i) = entI_{j_i}$, if entity is not abstract and has instances;
- $(\exists\, ent_i \in Ent) : fe(ent_i) = \varnothing$, if entity is abstract and does not have instances.

Map fe defines creation operation of node corresponding to entity instance.

Let's define map of metamodel graph nodes $EAttr$ conforming to a set of entity attributes on a set of model graph nodes $EAttrI$:

$$fea : EAttr \to EAttrI.$$

And besides

$$(\forall eattr_{j_i} \in EAttr)(\exists\, eattrI_{k_{j_i}} \in EAttrI):\ fea(eattr_{j_i}) = eattrI_{k_{j_i}},$$
$$i = \overline{1, |Ent|},\ j_i = \overline{1, |EntI_i|},\ k_{j_i} = \overline{1, |EAttrI_{j_i}|}.$$

Map fea corresponds to the operation of assignment a value to entity instance attribute.

Let's examine a set of metamodel graph nodes which correspond to relations. With each node we associate a set of graph model nodes that appropriate to particular relation instances, as a result we obtain a map $fr : Rel \to RelI$, such that the following

- $(\exists\, rel_i \in Rel)(\exists\, reli_{j_i} \in RelI):\ fr(rel_i) = reli_{j_i}$, if relation has instances;
- $(\exists\, rel_i \in Rel):\ fr(rel_i) = \varnothing$, if relation does not have instances.

This map defines creation operation of node corresponding to relation instance.

Let's define operation of assignment a value to relation instance attribute. To do this, we will construct a map of metamodel graph nodes $RAttr$ conforming to a set of relation attributes on set of model graph nodes corresponding to attribute values $RAttrI$: $fra : RAttr \to RAttrI$.

And besides

$$(\forall rattr_{j_i} \in RAttr)(\exists\, rattrI_{k_{j_i}} \in RAttrI):\ fra(rattr_{j_i}) = rattrI_{k_{j_i}},$$
$$i = \overline{1, |Rel|},\ j_i = \overline{1, |RelI_i|},\ k_{j_i} = \overline{1, |RAttrI_{j_i}|}.$$

Thus, maps fe, fea, fr, fra define matching between set of metamodel graph nodes and set of model graph nodes (Fig. 5).

Now we will define the rules under which the arcs of graph GMM are mapped to the arcs of graph GM.

Let's construct the map $gea : EEA \to EEAI$, according to which each arc of the set EEA is put in correspondence with specified arcs of the set $EEAI$, i.e.

$$(\forall eea_{j_i} \in EEA)(\exists\, eeaI_{k_{j_i}} \in EEAI):\ gea(eea_{j_i}) = eeaI_{k_{j_i}},$$
$$i = \overline{1, |Ent|},\ j_i = \overline{1, |EntI_i|},\ k_{j_i} = \overline{1, |EAttrI_{j_i}|}.$$

Similarly, we can define a map $gra : ERA \to ERAI$ for which

$$(\forall era_{j_i} \in ERA)(\exists\, eraI_{k_{j_i}} \in ERAI):\ gra(era_{j_i}) = eraI_{k_{j_i}},$$
$$i = \overline{1, |Rel|},\ j_i = \overline{1, |RelI_i|},\ k_{j_i} = \overline{1, |RAttrI_{j_i}|}.$$

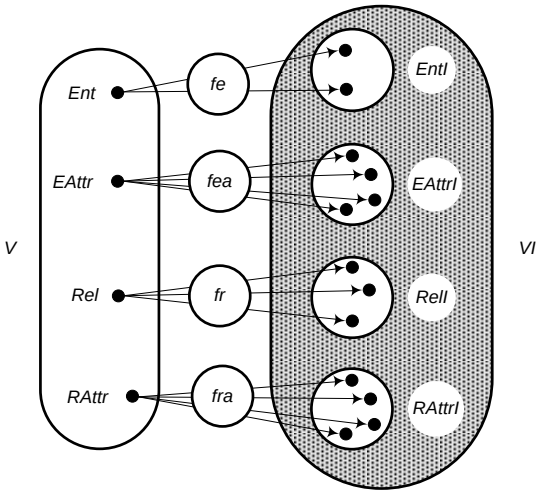


Figure 5. The map of metamodel graph nodes on model graph nodes

Let's construct the map $ger : EERR \to EERRI$, according to which each arc of the set $EERR$ is put in correspondence with specified arcs of the set $EERRI$, i.e.

$$(\forall eerr_i \in EERR)(\exists\, eerrI_{k_{j_i}} \in EERRI):\ ger(eerrI_{k_{j_i}}) = eerr_i,$$
$$i = \overline{1, |Ent|},\ j_i = \overline{1, |EntI_i|},\ k_{j_i} = \overline{1, |EAttrI_{j_i}|}.$$

Thus, maps gea, gra, ger define matching between the set of metamodel graph arcs and the set of model graph arcs.

Model graph creation is a map of metamodel graph on model graph at which conversions are performed $fe, fea, fr, fra, gea, gra, ger$.

D. Operation of Model Interpretation

Let's construct map of model graph on metamodel graph. It defines operation of model interpretation which allows to execute operations over entity instances and to check constraints imposed on the entities and relations.

As model graph nodes are instances of metamodel graph nodes, it is possible to define the map of the model graph nodes on the metamodel graph nodes.

Let's construct a surjection $fe^{-1} : EntI \to Ent$ which to each model entity instance puts in correspondence metamodel entity

$$(\forall entI_{j_i} \in EntI)(\exists!\, ent_i \in Ent):\ fe^{-1}(entI_{j_i}) = ent_i,$$
$$i = \overline{1, |Ent|},\ j_i = \overline{1, |EntI_i|},$$

and besides several elements of the set $EntI$ may correspond to one entity, i.e. is performed

$$(\forall ent_i \in Ent)(\exists\, entI_{j_i}, entI_{k_i} \in EntI, entI_{j_i} \neq entI_{k_i}):$$
$$fe^{-1}(entI_{j_i}) = fe^{-1}(entI_{k_i}) = ent_i.$$

Let's define map which is an inverse of map fea:

$$fea^{-1} : EAttrI \to EAttr .$$

This surjection to each element of set $EAttrI$ puts in correspondence a unique element of set $EAttr$, i.e.

$$(\forall eattrI_{k_{ji}} \in EAttrI)(\exists ! eattr_{j_i} \in EAttr) : fea^{-1}(eattrI_{k_{ji}}) =$$
$$= eattr_{j_i}, i = \overline{1, |Ent|}, j_i = \overline{1, |EntI_i|}, k_{j_i} = \overline{1, |EAttrI_{j_i}|},$$

and besides several elements of the set $EAttrI$ may correspond to one element of the set $EAttr$, i.e. is performed

$$(\forall ea_{j_i} \in EAttr)(\exists\, eaI_{k_{ji}}, eaI_{l_{ji}} \in EAttrI, eaI_{k_{ji}} \neq eaI_{l_{ji}}) :$$
$$fea^{-1}(eaI_{k_{ji}}) = fea^{-1}(eaI_{l_{ji}}) = ea_{j_i} .$$

Let's consider a set of model graph nodes that correspond to relation instances. Each such node we associate with a unique metamodel graph node, which corresponds to a current relation, as a result we obtain a surjective map $fr^{-1} : RelI \to Rel$ for which

$$(\forall relI_{j_i} \in RelI)(\exists ! rel_i \in Rel) : fr^{-1}(relI_{j_i}) = rel_i,$$
$$i = \overline{1, |Rel|}, j_i = \overline{1, |RelI_i|},$$

and multiple relation instances may be created on the basis of one relation, i.e. is performed

$$(\forall rel_i \in Rel)(\exists\, relI_{j_i}, relI_{k_i} \in RelI, relI_{j_i} \neq relI_{k_i}) :$$
$$fr^{-1}(relI_{j_i}) = fr^{-1}(relI_{k_i}) = rel_i .$$

Surjective map $fra^{-1} : RAttrI \to RAttr$ which is an inverse of map fra, each model node conforming to relation attribute value associates with a unique metamodel node from set $RAttr$:

$$(\forall rattrI_{k_{ji}} \in RAttrI)(\exists ! rattr_{j_i} \in RAttr) : fra^{-1}(rattrI_{k_{ji}}) =$$
$$= rattr_{j_i}, i = \overline{1, |Rel|}, j_i = \overline{1, |RelI_i|}, k_{j_i} = \overline{1, |RAttrI_{j_i}|},$$

and multiple elements of the set $RAttrI$ may correspond to one element of set $RAttr$, i.e. is performed

$$(\forall ra_{j_i} \in RAttr)(\exists\, raI_{k_{ji}}, raI_{l_{ji}} \in RAttrI, raI_{k_{ji}} \neq raI_{l_{ji}}) :$$
$$fra^{-1}(raI_{k_{ji}}) = fra^{-1}(raI_{l_{ji}}) = ra_{j_i} .$$

Thus, four maps fe^{-1}, fea^{-1}, fr^{-1}, fra^{-1} define matching between the set of model graph nodes and the set of metamodel graph nodes (see Fig. 6).

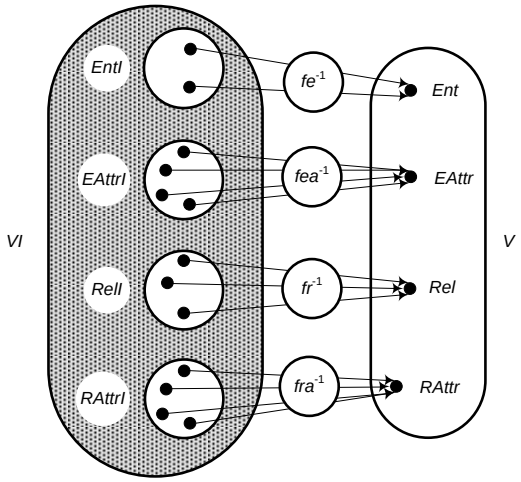


Figure 6. The map of model graph nodes on metamodel graph nodes

Since operations over entity and relation instances are not defined, then for navigation between the entities, relations and their instances let's extend set of model graph arcs with the arc-references connecting entity and relation instances with those metamodel entities and relations on which basis they are created. Let's denote the set of such arcs through

$$T = \bigcup_{i=1}^{|Ent|+|Rel|} T_i, T_i = \{t_{j_i}\}, j = \overline{1, |EntI_i| + |RelI_i|} .$$

Now we will define the rules under which the arcs of model graph GM are associated with the arcs of metamodel graph GMM.

Let's construct the map $gea^{-1} : EEAI \to EEA$ which to each arc of the set $EEAI$ puts in correspondence unique arc of the set EEA, i.e.

$$(\forall eeaI_{k_{ji}} \in EEAI)(\exists ! eea_{j_i} \in EEA) : gea^{-1}(eeaI_{k_{ji}}) = eea_{j_i},$$
$$i = \overline{1, |Ent|}, j_i = \overline{1, |EntI_i|}, k_{j_i} = \overline{1, |AttrI_{j_i}|} .$$

Similarly, we can define a map $gra^{-1} : ERAI \to ERA$ for which

$$(\forall eraI_{k_{ji}} \in ERAI)(\exists ! era_{j_i} \in ERA) : gra^{-1}(eraI_{k_{ji}}) = era_{j_i},$$
$$i = \overline{1, |Rel|}, j_i = \overline{1, |RelI_i|}, k_{j_i} = \overline{1, |AttrI_{j_i}|} .$$

As can be seen from definition the maps gea^{-1} and gra^{-1} are surjective.

Let's construct the surjective map $ger^{-1} : EERRI \to EERR$ which to each arc of the set $EERRI$ puts in correspondence unique arc of the set $EERR$, i.e.

$$(\forall eerrI_{j_i} \in EERRI)(\exists ! eerr_i \in EERR) : ger^{-1}(eerrI_{j_i}) = eerr_i,$$
$$i = \overline{1, |EERR|}, j_i = \overline{1, |EERRI_i|} .$$

Thus, maps $gea^{-1}, gra^{-1}, ger^{-1}$ define single-valued transformation between set of model graph arcs and set of metamodel graph arcs.

Model interpretation is a map of model graph on metamodel graph at which conversions are performed $fe^{-1}, fea^{-1}, fr^{-1}, fra^{-1}, gea^{-1}, gra^{-1}, ger^{-1}$.

V. Development Environment of Metalanguage System

To work with metalanguage objects the development environment that includes the following components: graphical editor, object browser, repository, validator, generator is designed.

The *development environment* includes implementation of the general service functions of created system. It integrates all components into a single unit.

Graphic Editor – a work area for drawing diagrams. Assignment of the Editor is a creation, modification, removal of models, and also establishment of links between different models. Each model entity is represented by some graphic symbol, and relations between entities are represented by different types of lines.

The Graphical Editor allows to allocate on a worksheet various shapes (instances of entities and relations), to apply to these shapes different actions, to set various graphical properties for them.

Object Browser – a tool designed for viewing and editing information stored in the repository. The browser provides the ability to export/import models to/from external systems. A format for models import/export is the XML which contains besides the data also metadata that describe structure of the stored information.

Uniform storage of all information about the system is the *repository*. It contains the information about metamodels, models, entities, relations, attributes, constraints, icons used to image entities and relations. Repository stores the information about models as well as metamodels uniformly it allows to process them with a single tool. Physically, the repository is a relational database.

The *Validator* checks correspondence of model to the constraints specified by the user. At check each constraint will be applied to each instance of entities and relations. If constraint is not performed, the error message will be shown.

The *Generator* allows generating XML-file, model documentation or source code on the basis of existing models. XML-file will contain information about the model: model properties, entities, relations, their attributes, constraints imposed on the model. Model documentation includes: model name, information about developers who took part in its creation, graphical representation of model with links to description of its individual parts.

Having described the basic components of a MetaLanguage system, let consider how visual domain-specific modeling languages are designed (Fig. 7).

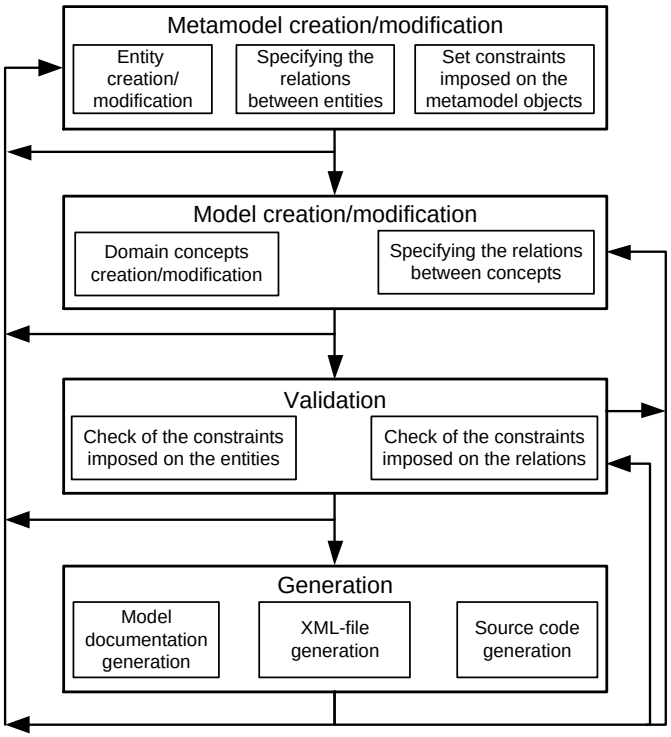


Figure 7. Process of creation/modification the by means of MetaLanguage system

Process of DSL definition begins with metamodel creation. For this purpose it is necessary to specify the main constructions of created language, to define relations between them, to set constraints imposed on the metamodel entities and relations. After building of metamodel the developer gets a customizable extensible visual modeling language.

Using created DSL, the user can design models containing objects that describe specific domain concepts and links between them.

The Validator should check up whether model satisfies to constraints which were imposed on it after model constructing.

Using the Generator, the developer can save the constructed metamodels and models in the form of XML-files or generate system documentation or source code based on them.

Note that at metamodel modification the system automatically will make all necessary changes in the models which are created on the basis of this metamodel.

VI. Conclusion and Future Works

The article describes the language workbench MetaLanguage which can be used at all stages of information system creation from domain-specific modeling languages development to creating of models that used in a particular system implementation or for source code generation.

The analysis of existing analogues has shown that there are unresolved problems: impossibility of export of DSLs and models to external systems, impossibility of models

transformations from one notation to another, impossibility of dynamic adaptability of languages. It was decided to eliminate these DSM-platforms restrictions at MetaLanguage system engineering.

The development environment is simple to use, therefore not only professional programmers, but also domain experts, for example, business analysts, can work with this toolkits. Thus the developer gets powerful workbench for creation of visual dynamic adaptable domain-specific modeling languages.

To work with models and metamodels uniformly, it is used the same tools, therefore process of model creation can be iterative.

Metamodels modification can be made at any stage of DSL creation. Thus after metamodel modification the system automatically will make all necessary changes in models which are created on basis of this metamodel.

For unified models creation the mathematical model – graph grammars based on pseudo-metagraphs – was constructed. This formalism has allowed to describe basic elements and algorithms which MetaLanguage uses in its work: algorithms for creation/modification of domain metamodels and models, algorithms for vertical models transformation, algorithms for constraint checking.

The paper also presents the approaches to implementation of metalanguage and development environment to work with it. This environment allows to create modeling languages that

- can be flexibly configured not only to ever-changing needs of business processes and users, but also to other domains;
- provide an opportunity to work in domain terms;
- have a high degree of consistency with the metalanguage;
- can be reused in similar projects.

The research prototype of MetaLanguage system that implements the functionality described above was created in the present time. In the future it is planned to continue working in this direction:

- to design the DSLs for various purposes, for example, for description of ontologies, document templates, business processes with created DSM-platform;
- to describe algorithms for the horizontal transformation of graph representation which will allow to make transformation of domain models from one notation to another;
- to integrate language workbench MetaLanguage with some CASE tool that allows to develop information systems, for example, with METAS CASE system [19].

REFERENCES

[1] А.О. Сухов, Л.Н. Лядова "Использование визуальных предметно-ориентированных языков для описания бизнес-процессов", Материалы межвуз. конкурса-конференции "Технологии Microsoft в теории и практике программирования", СПб, 2009. С. 117.

[2] А.О. Сухов "Использование предметно-ориентированных языков при создании приложений для мобильных устройств", Материалы всероссийской научно-практической конференции студентов "Студент и наука", т. 3, 2010. С. 75-76.

[3] M. Fowler, “Language Workbenches: The Killer-App for Domain Specific Languages?” Available at: http://martinfowler.com/articles/languageWorkbench.html (accessed 10 April 2012).

[4] Л.Н. Лядова, А.О. Сухов "Визуальные языки и языковые инструментарии: методы и средства реализации", Труды международных научно-технических конференций "Интеллектуальные системы" (AIS’10) и "Интеллектуальные САПР" (CAD-2010), т. 1, 2010. С. 374-382.

[5] В.С. Маторин "CASE-инструментарий UFO-toolkit. Автоматизация построения УФО-моделей", Проблемы программирования, №2, 2004. – С. 144-149.

[6] А.Н. Иванов "Технологическое решение REAL-IT: создание информационных систем на основе визуального моделирования", Сб. "Системное программирование" под ред. проф. А.Н.Терехова и Д.Ю.Булычева, 2004. – С.89-100.

[7] J.-P. Tolvanen, M. Rossi, “MetaEdit+: defining and using domain-specific modeling languages and code generators.” Available at: http://portal.acm.org/citation.cfm?id=949365 (accessed 10 April 2012).

[8] S. Cook, G. Jones, S. Kent, A.C. Wills, “Domain-Specific Development with Visual Studio DSL Tools,” Reading: Addison-Wesley, 2007.

[9] “Creating Domain-Specific Languages.” Available at: http://msdn.microsoft.com/en-us/library/bb126259(v=vs.80).aspx (accessed 10 April 2012).

[10] R.C. Gronback “Eclipse Modeling Project: A Domain-Specific Language Toolkit,” Reading: Addison-Wesley, 2009.

[11] T. Ozgur, “Comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks for Domain-Specific Modeling In the context of the Model-Driven Development.” Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.6383&rep=rep1&type=pdf (accessed 10 April 2012).

[12] А.В. Ларионов "Разработка визуального языка автоматного программирования". Available at: http://is.ifmo.ru/papers/StateMachineDesigner.pdf (accessed 10 April 2012).

[13] S. Dmitriev, “Language Oriented Programming: The Next Programming Paradigm.” Available at: http://www.onboard.jetbrains.com/is1/articles/04/10/lop/index.html (accessed 10 April 2012).

[14] А.П. Стасенко "Автоматная модель визуального описания синтаксического разбора", Вычислительные технологии, вып. 5, т. 13, 2008. – С. 70-87.

[15] О.Ф. Королев "Алгоритмические сети как визуальный язык программирования", Труды СПИИРАН, вып. 2, 2005. – С. 130-137.

[16] J. Rekers, A. Schuerr, “A Graph Grammar approach to Graphical Parsing,” Visual Languages Proceedings, 11th IEEE International Symposium, Darmstadt, pp. 195-202, 1995.

[17] B. Courcelle, “Graph Rewriting: An Algebraic and Logic Approach,” Handbook of Theoretical Computer Science, vol. B, 1990, pp. 193-242.

[18] А.О. Сухов "Анализ формализмов описания визуальных языков моделирования", Современные проблемы науки и образования, №2, 2012. Available at: http://www.science-education.ru/102-5655 (accessed 10 April 2012).

[19] Л.Н. Лядова, С.А. Рыжков "CASE-технология METAS", Математика программных систем, Межвуз. сб. науч. статей, Пермь, 2003. С. 4-18.

New Developments of the Computer Language Classification Knowledge Portal

Aleksandr Akinin
Novosibirsk State University,
Novosibirsk, Russia
Email: akinin3113@gmail.com

Alexey Zubkov
Novosibirsk State University
Novosibirsk, Russia
Email: ortoslon@gmail.com

Nikolay Shilov
Institute of Informatics Systems,
Novosibirsk, Russia
Email: shilov@iis.nsk.su

***Abstract*—During the semicentennial history of Computer Science and Information Technologies, several thousands of computer languages have been created. The computer language universe includes languages for different purposes (programming, specification, modeling, etc.). In each of these branches of computer languages it is possible to track several approaches (imperative, declarative, object-oriented, etc.), disciplines of processing (sequential, non-deterministic, distributed, etc.), and formalized models, such as Turing machines or logic inference machines. Computer language paradigms are the basis for classification of the computer languages. They are based on joint attributes which allow us to differentiate branches in the computer language universe. We have presented our computer-aided approach to the problem of computer language classification and paradigm identification in a recent paper *Development of the Computer Language Classification Portal* (Proc. of Ershov Informatics Conference PSI-2011, Lect. Not. in Comp. Sci., v.7162). In the present paper we discuss new developments of our project: (1) the pre-alpha version of the Portal is online, and (2) the reasoner is a model checking engine for a paraconsistent (inconsistency-tolerant) description logic.**

***Keywords*: computer languages, computer paradigm, classification, knowledge portal, description logic, paraconsistency, inconsistency-tolerance, model checking.**

I. Introduction: The Problem of Computer Language Classification

Let us start with a sketch of motivation for our research. Please refer [12] for more details.

We understand by a computer language any language that is designed or used for automatic information processing, i.e. data and process representation, handling and management. A classification of some universe (the universe of computer languages in particular) consists in means of identification and separation of items/entities/objects, classes and their roles, and navigation between them.

The *History of Programming Languages* poster by O'REILLY is well known [15]. It represents chronological and influence relations between 2500 programming languages. Due to the number of existing computer languages alone, there is a necessity for their systematization or, more precisely, for their classification. At the same time, classification of already developed and new computer languages is a very important problem for Computer Science, since software engineers and information technology experts could benefit by a sound framework for computer language choice of components for new program and information systems.

Drawing an analogy between Computer Science and other sciences, one may assume that classification of computer languages could be done in the style of Linnaeus (i.e., a taxonomy like: Kingdom - Phylum - Class - Order - Family - Subfamily - Genus - Species). For example, look at *Taxonomic system for computer languages* [17].

However, there is a great difference between domains of natural sciences and Computer Science since the former is static while the latter is highly dynamic. In the last decade of the twentieth century everyone can see rapid growth of existing and new branches of computer languages (knowledge representation languages, languages for parallel/concurrent computing, languages for distributed and multi-agent systems, etc.). Each of these new computer languages has its own, sometimes very particular syntax, a certain model of information processing (i.e., semantics or a virtual machine), and its pragmatics (i.e., the sphere of its application and distribution). And though there were rather small groups of computer languages (e.g., Hardware Description Languages), many groups had already been crowded (e.g., Specification Languages) and some of them went through the period of explosion and migration (e.g., Markup Languages). Sometimes computer language experts have difficulties in putting some languages into one definite group. For example, the programming language Ruby: "Its creator, Yukihiro "matz", blended parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp) to form a new language that balanced functional programming with imperative programming"[18]. Rapid generation of new computer languages will continue while new spheres of human activities will be computerized.

We think that a modern classification of the computer languages universe can be built upon the flexible notion of *computer language paradigms*. In the general methodology of science, *paradigm* is an approach to the formulation of problems and their solutions. The contemporary meaning of the term is due to the well-known book [5] by Thomas Kuhn. Robert Floyd was the first who had explicitly used the term

"paradigm" in the Computer Science context. In particular, he addressed "Paradigms of Programming" in his Turing Award Lecture [3]. Unfortunately, R. Floyd did not define this concept explicitly.

Recently Peter van Roy has published the taxonomy *The principal programming paradigms* [19] with 27 different paradigms and advocated it in the paper [10]. Surprisingly, the cited paper does not provide a convincing and concise definition of the notion *Programming Paradigm*. We can refer to the following quotation only: "A programming paradigm is an approach to programming a computer based on a mathematical theory or a coherent set of principles. Each paradigm supports a set of concepts that makes it the best for a certain kind of problem." [10]

In our recent paper [12] we suggested more comprehensive definition for computer paradigm that (we believe) is coherent with the general concept of *paradigm*:

1) Computer paradigms are alternative approaches (patterns) to formalization of information problem formulation, presentation, handling and processing.
2) They are fixed in the form of formal (mathematical) theory and accumulated in computer languages.
3) Every natural class of computer languages is the extent of some paradigm, and vice versa, every computer paradigm is the intent of some class of computer languages.
4) A paradigm can be characterized by a set of problems/application areas that the paradigm fits better than the other ones.
5) The educational value of paradigms is to teach to think different about information problems and to choose the best paradigm to solve them.

II. Methodology: the Syntactic-Semantic-Pragmatic Approach

Categories *syntax*, *semantics* and *pragmatics* are used to characterize natural and artificial languages (including computer languages). Syntax is the orthography of the language. The meaning of syntactically correct constructs is provided through language semantics. Pragmatics is the practice of use of meaningful, syntactically correct constructs. Therefore the approach that is based on features of syntax, semantics and pragmatics could be natural for specification of paradigms and classification of computer languages.

The syntactic aspect of computer language classification should reflect both the formal syntax and the human perspective. Certainly, it is very important for the compiler implementation whether a particular language has regular, context-free or context-sensitive syntax. Thus, syntactic properties of computer languages could be attributes in the classification. These attributes can be brought from formal language theory. But informal annotations (attributes) like *flexibility*, *naturalness*, *style* (supported by a library of good style examples), *clarity* from a human standpoint (including a portion of *syntactic sugar*) become much more important.

The role of semantics for computer languages is well known. But there are several problems with the use of *formalized semantics* in classification of computer languages, the major problems are listed below.

- Poor acquaintance with formal semantics among computer languages users, *more experts, but fewer general users*.
- Prejudice that formal semantics is too *pure in theory* but too *poor in practice*.
- Too many individual semantic systems and notations with different level of formalization are adopted for different computer languages.

Nevertheless, we think that these problems can be solved by development of multidimensional stratification of "paradigmatic" computer languages[1].

For example, educational semantics and formal semantics are two particular semantic dimensions. They can be stratified into *levels* and *layers* as follows.

- The layer hierarchy is an educational, human-centric semantic representation. It should comprise 2-3 layers that could be called *elementary*, *basic*, and *full*. The elementary layer may be an educational dialect of the language for the first-time study of primary concepts and features. The basic layer may be a subset for regular users of the language which requires skills and experience. The full layer is the language itself, it is for advanced and experienced users.
- The level hierarchy is a formal-oriented semantic representation. It should comprise several levels for the basic layer of the language and optionally for some other layers. The levels of the basic layer could be called *kernel*, *intermediate*, and *complete*. The kernel level would have executable semantics and provide tools for the implementation of the intermediate level; the intermediate level in turn should provide implementation tools for the complete level. Implementation of intermediate level should be of semantics-preserving transformation. Please refer to [9] for an example of a three-level hierarchy for the programming language C#.

In contrast to syntax and semantics, pragmatics relies upon highly informal *beliefs* (i.e. expertise and experience) of people that are involved in the computer language life cycle (i. e. design, implementation, promotion, usage and evolution). In other words, we need to represent formally *expert "knowledge"* (i.e. their views and beliefs) about computer languages, related concepts, and relations between computer languages. It naturally leads to the idea of representing this "knowledge" with an ontology. It is just a tradition to call experts' *beliefs knowledge*, since this expertise can be just an authoritative opinion, but not true, while (according to Plato) *knowledge is true belief*. Nevertheless we will follow this tradition in spite of inconsistency with epistemology.

[1] *Paradigmatic languages* are the most typical ones for a particular paradigm (class).

Formal "ontology is the theory of objects and their ties. Ontology provides criteria for distinguishing various types of objects (concrete and abstract, existent and non-existent, real and ideal, independent and dependent) and their ties (relations, dependencies and predication)" [20]. A formal ontology (simply *ontology* in the sequel) of a particular *problem domain* is a formalization of knowledge about objects (entities) of the domain (computer languages for instance), their classes and ties (relations). This knowledge could include empirical facts, mathematical theorems, personal beliefs, etc.

Expert knowledge for pragmatics of computer languages should be formalized in an *open*, *evolving* (i.e. versioned and temporal) ontology that includes syntactic and semantic (both formal and informal) knowledge in the form of annotations and attributes. The *openness* means that the ontology is open for access and editing. Temporality means that the ontology changes in time, admits temporal queries and assertions, and that all entries in the ontology are timestamped. Versioning means that the ontology tracks all its changes. Wikipedia, the free encyclopedia, is a good example an of open and evolving ontology.

III. Towards an Open Temporal Evolving Ontology for the Classification of Computer Languages

A. Existing Ontologies of Programming Languages

History of Programming Languages poster by O'REILLY [15] can be considered as a primitive ontology of programming languages that is neither open nor evolving. Programming languages are the objects in this ontology, but, unfortunately, the poster does not provide any information about classes of objects. The navigation method in this ontology is represented by *influence lines* and *chronology*.

History of Programming Languages (HOPL) [16] is a much better-developed ontology of programming languages, but, unfortunately, it is, too, neither open for editing nor evolving. HOPL represents historical and implementation information about an impressive number (>8500) of programming languages, but hasn't been updated since 2006, and does not deal with any inter-language relations other than *language-dialect-variant-implementation*.

The situation is different with *Progopedia* [21], a wiki-like encyclopedia of programming languages. It is open for editing and is tracing its history. But Progopedia has poor temporal navigation means. While HOPL provides some taxonomy instruments, Progopedia only has a trivial one *language-dialect-variant-implementation*. In comparison with HOPL and the O'REILLY poster, Progopedia is relatively small. At present it contains information about ∼130 languages, ∼70 dialects, ∼300 implementations, and ∼660 versions.

None of the three listed ontologies have means for constructing classes by users or deriving classes, and only manual navigation among the classes is supported. We believe that a more comprehensive ontology is needed to solve the problem of computer languages classification, i.e. identification and differentiation of classes of computer languages and navigation among them.

B. Outlines of our Approach

We develop ontology for computer languages, based on Description Logic (DL) [1], [11], [14]. The objects of our ontology are computer languages (also their levels and layers), concepts/classes (in terms of DL/OWL) — collections of computer languages that can be specified by concept terms (in DL terms), ties (DL-roles or OWL-properties) — relations between computer languages. For example, Pascal, LISP, PROLOG, SDL, LOTOS, UMLT, as well as C, C-light and C-kernel, OWL-Lite, OWL-DL and OWL-full should eventually become objects of the ontology.

Since we understand computer paradigms as specifications of classes of computer languages, and we consider classes of computer languages as DL-concepts (OWL-classes), then we have to adopt DL concepts as paradigms of computer languages: *Every (syntactically correct) DL concept term defines a paradigm that is the concept specified by the term*. In this setting, computer language paradigms and classification is not a taxonomic tree based on property inheritance from sup-class to sub-class, but a formal ontology with navigation by DL means.

Objects (i.e. computer languages) of the ontology could be described with different formal attributes (e.g., formal syntax properties) and informal annotations (e.g., libraries of samples of good style). Let us remark that the list of formal attributes and informal annotations is not fixed but is open for modifications and extensions. Nevertheless, we fix certain attributes and annotations for all objects (but allow to assign an indefinite value for them). For example, we provide the following attributes:

- *date of birth* with various time granularity,
- *URL of an external link* for any non-specified references,
- *try-version* for a link to an easy to install or web-based small implementation (that can be freeware or shareware).

Some elementary concepts/classes in the ontology are also fixed, for example: *has context-free syntax*, *functional languages*, *specification languages*, *executable languages*, *static typing*, *dynamic binding*, etc. A special elementary concept/class is *paradigmatic computer languages*, it comprises few (but one at least) representatives for every elementary concept/class. We expect to borrow more ideas for elementary concepts from [22]. Elements of elementary concepts/classes must be explicitly annotated by appropriate attributes (*has a context-free syntax*, *is a functional language*, *is a specification language*, etc.).

Non-elementary concepts/classes should be specified by DL concept terms. For example, *executable specification languages* is the intersection of *executable languages* and *specification languages*. Since our ontology is an open-world ontology with incomplete information then some problem occurs with *class-complement*. For example, if a language has no explicitly attached attribute *has a context-free syntax*, it does not mean that the language has no CF-syntax, it just

means that *the information is not available*. To resolve the problem, we provide every *positive* attribute (e.g., *has context-free syntax*) by the corresponding *negative* attribute that is the counterpart of positive one (e.g., *DOES NOT have a context-free syntax*).

All elementary concepts/classes (including paradigmatic languages) should be created on the basis of expert knowledge and be open for editing. A special requirement for the proposed ontology should be the following constraint: every legal (i.e. *well-formed*) non-empty concept/class must contain a paradigmatic language. This is common sense: if experts can not point out a representative example of a paradigm, then it should be empty.

Roles/properties in the proposed ontology could also be natural: *is a dialect of*, *is a layer of*, *uses the syntax of*, etc. For example: *C-light is a layer of C*, *OWL uses the syntax of XML*, etc. All listed examples are elementary DL-roles/OWL-properties. Standard (positive) relational algebra operations *union*, *intersection*, *composition*, *role inverse*, and *transitive closure* can be used and are meaningful for construction of new roles/properties. For example, *uses the syntax of a dialect of* is the composition of *uses the syntax of* and *is a dialect of*. Again we have a problem with *role complement*, but we have not fix any solution yet (in contrast to the *class-compliment* problem).

Let us remark that the computer language domain has four domain-specific ties between languages: *is a dialect of*, *is a variant of*, *is a version of*, and *is an implementation of*. Of course these ties must be present in the proposed ontology as elementary DL-roles/OWL-properties. But, unfortunately, there is no consensus about definition of these ties. For example, *Progopedia* [21] considers that an implementation can have a version, while [22] promotes an opposite view that a version can have an implementation. Currently we adopt the following definition.

- *Dialects* are languages with joint elementary level.
- *Variants* are languages with joint basic level.
- *Version series* is a partially ordered collection of variants such that every smaller version is a compatible subset of all later versions.
- *Implementation* is a platform-dependent variant of a language.

Let us remark that several incompatible versions can coexist:Object C and C++ are object-oriented variants of C, but for sure these two languages are incompatible.

Universal and *existential* quantifier restrictions that are used in OWL and DL for construction of new classes/concepts have a natural and useful meaning. An example of existential restriction (in DL notation): a concept $(markup\_language) \sqcap \exists uses\_syntaxof : (\neg\{XML\})$ consists of all computer languages that are markup languages but do not use the syntax of the Extensible Markup Language XML; an example of a language of this kind is LaTeX. An example of a universal restriction and a terminological sentence (in DL notation also) follows: the sentence $\{XML\} \sqsubseteq is\_dialect\_of : (\neg\{ML\})$ expresses that XML is a dialect of any computer language but the functional programming language ML.

IV. Current State of the Project

We started implementation of a prototype of a computer languages classification knowledge portal (that eventually will evolve into an open temporal evolving ontology) for classification of computer languages a year ago [12]. At present, a pre-alpha version of the portal is available online [23].

The prototype does not support full functionality. The prototype is implemented as a web application, so everyone can enter it with a web browser. The interface allows users to view and edit information contained in the portal, which is formed as an ontology.

The main elements of the prototype ontology are computer languages (objects of the ontology), elementary classes of languages (arbitrary, explicitly user-specified subsets of the set of objects), relations between the languages (binary relations over the set of objects), attributes (mappings from the set of languages to some external data types, e.g. text strings, URL's) and the Knowledge Base (Description Logic statements that represent laws of the problem domain of Computer Languages). The data is represented internally as an RDF repository. All these entities can be viewed and modified directly by the user.

Two main services (that are already provided) are the ontology model checker and visualization. The model checker is used for computing classes of objects and ties from specifications (concept and role terms), and for checking consistency of the ontology (data and the Knowledge Base). Visualization is used for displaying classes and ties graphically.

The model checker is an explicit-state model checker for a *paraconsistent* (i.e. inconsistency-tolerant) *description logic* [6], [7], [8] extended by two special constructs for concept terms borrowed from Formal Concept Analysis (FCA) [4], [14], [11]. The underlying paraconsistent description logic uses four-value semantics of Belnap logic [2]. The constructs borrowed from FCA are upper and lower derivatives. (The lower derivative is the same as the *window operator* in DL.) The logic is chosen to handle openness of the ontology and incompleteness and inconsistency of data in the ontology.

Why do we use a model checker as a reasoning tool instead of any available DL inference machine (such as Fact++, Kaon2, etc.)? Because our ontology is for empirical expert knowledge about rapidly developing and changing domain of Computer Languages, not a domain with a set of predefined domain-specific laws. We use an *explicit-state* model checker (not a *symbolic* one) since the domain numbers thousands of objects, i.e. it fits explicit-state representation well.

Why are we developing a self-contained tool for the ontology instead of using some other ontology tool (Protege for instance)? Because we are developing a tool for a small community-oriented ontology for Computer Language experts, where people would like to use a simple interface instead of studying a manual or a tutorial before using the tool.

We would like to emphasize that at present the ontology is an open ontology already. We expect that the ontology eventually will also become versioned and evolving, i.e. will support automatic timestamping, history of all edits, and temporal queries. We would like to hope that our ontology and portal will provide researchers by a sound and easy framework for language specification as well as software engineers and IT managers by tools for language choice.

ACKNOWLEDGMENT

Research is supported by Integration Research Program n.3 (2012-2013) *Ontology Design and Development on base of Conceptualization by means of Logic Description Languages* provided by Siberian Branch, Russian Academy of Science.

REFERENCES

[1] F. Baader, D. Calvanese, D. Nardi, D. McGuinness, and P. Patel-Schneider (editors), *The Description Logic Handbook: Theory,Implementation and Applications*, Cambridge University Press, 2003.

[2] N.D. Belnap, *How a computer should think*, Contemporary Aspects of Philosophy: Proceedings of the Oxford International Symposium, 1977, p.30-56.

[3] R.W. Floyd, *The paradigms of Programming*, Communications of ACM, v.22, 1979, p.455-460.

[4] B. Ganter, R. Wille, *Formal Concept Analysis. Mathematical Foundations*, Springer Verlag, 1996.

[5] T.S. Kuhn, *The structure of Scientific Revolutions*, Univ. of Chicago Press, 1970. (3rd Ed. – 1996.)

[6] Y. Ma, P. Hitzler, and Z. Lin, *Algorithms for paraconsistent reasoning with owl*, Proc. of European Semantic Web Conference 2007, Lect. Not. in Comp. Sci., v.4519, Springer, 2007, p.399-413.

[7] Y. Ma, P. Hitzler, and Z. Lin, *Paraconsistent reasoning for expressive and tractable description logics*, Proc. of the 21st International Workshop on Description Logic, CEUR Electronic Workshop Proceedings, v.353, 2008.

[8] Y. Ma and P. Hitzler, *Paraconsistent Reasoning for OWL 2*, Proc. of Web Reasoning and Rule Systems, Lect. Not. in Comp. Sci., v.5837, 2009, p.197-211.

[9] V.A. Nepomniaschy, I.S. Anureev, I.V. Dubranovskii, and A.V. Promsky, *Towards verification of C# programs: A three-level approach*, Programming and Computer Software, v.32(4), 2006, p.190-202.

[10] P. van Roy, *Programming Paradigms for Dummies: What Every Programmer Should Know*, in New Computational Paradigms for Computer Music, IRCAM/Delatour, France, 2009, p.9-38.

[11] N.V. Shilov, *Formal Models and Methods for Design and Development of Ontologies*, in Formal Logical and Linguistic Models and Methods for Design and Development of Information Systems, Siberian Devision of Russian Academy of Sciences, 2009, p.11-48 (in Russian).

[12] N.V. Shilov, A.A. Akinin, A.V. Zubkov, and R.I. Idrisov, *Development of the Computer Language Classification Portal*, Proc. of Ershov Informatics Conference, Lect. Not. in Comp. Sci., v.7162, 2012, p.340-348.

[13] N.V. Shilov, *Make Formal Semantics Popular and Useful*, Bulletin of the Novosibirsk Computing Center (Series: Computer Science, IIS Special Issue), v.32, 2011, p.107-126.

[14] S. Staab and R. Studer (editors) *Handbook on Ontologies*, International Handbooks on Information Systems. Springer, 2nd edition, 2009.

[15] *History of Programming Languages*, available at `http://oreilly.com/news/graphics/prog_lang_poster.pdf`.

[16] *History of Programming Languages*, available at `http://hopl.murdoch.edu.au/`.

[17] *Taxonomic system for computer languages*, available at `http://hopl.murdoch.edu.au/taxonomy.html`.

[18] *Ruby. A Programmer's best friend*, available at `http://www.ruby-lang.org/en/about/`.

[19] *The principal programming paradigms*, available at `http://www.info.ucl.ac.be/~pvr/paradigms.html`.

[20] *Ontology. A Resource Guide for Philosophers*, available at `http://www.formalontology.it/`.

[21] *Progopedia*, available at `http://progopedia.ru/`.

[22] *The Language List*, available at `http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm`.

[23] *Computer Language Classification*, available at `http://complang.somee.com/Default.aspx`.

Generating Test Cases With High Branch Coverage for Web Applications

Andrey Zakonov and Anatoly Shalyto
National Research University of Information Technologies,
Mechanics and Optics, Saint-Petersburg, Russia
Email: andrew.zakonov@gmail.com, shalyto@mail.ifmo.ru

***Abstract*—Web applications have become significantly more complex and have begun to be used in wide variety of areas including social networks, shopping, online banking and other safety critical systems. We present an approach for automated white-box test generation for web applications. Our approach is to convert problem of high branch coverage test suite generation into a reachability problem and to utilize existing software verification techniques to generate test data for each execution branch. Set of Abstract Syntax Trees (ASTs) is built that describes web application as a whole, both client- and server-side code, by analyzing JavaScript code, its AJAX server calls and callbacks. Constructed AST is converted into an C# function with similar behavior and a set of arguments that represent user inputs. Existing test automation tools are used to discover test data that covers all the possible execution branches in a C# function. Web application test cases are generated with the discovered values, Selenium toolkit is used to emulate user actions and to automatically run the program under test against test cases.**

I. Introduction

Over the recent years web applications have begun to be used in wide variety of areas and have become safety critical, as often contain personal or financial information. Nevertheless relatively little tool support is available to assist testing of web applications. Web applications are usually tested by manually constructed test cases using unit testing tools with capture-replay facilities such as Selenium [1] and Sahi [2]. These tools provide functionality for recording the GUI actions performed by a user in test scripts, for running a suite of tests, and for visualizing test results. However, even with such tool support, testing remains a challenging and time-consuming activity because each test case has to be constructed manually.

As a web application we consider a set of pages connected by hyperlinks, ranging from a set of html pages displaying static content to a complex single page applications, which Document Object Model (DOM) could be dynamically modified by the application logic implemented as JavaScript handlers for user interactions and AJAX callbacks. Originally designed for simple scripting, modern JavaScript programs are complex pieces of software that interact with users and servers and play central role in a web application.

Because of web applications' client-server model, asynchronous server communication and event-driven nature tools and approaches developed for structured and object oriented programs can not be applied out-of-the box for web applications quality assurance. In [4] we addressed problem of creating models of web applications and proposed to use random-driven approach to cover different web application states. In [3] a method to apply Genetic Algorithms to test generation problem for a given EFSM was proposed. In the current research we develop an approach that automates process of test cases generation for a given web application state and generates a test suite with high branch coverage.

There are approaches that generate test cases for structured programs with high branch coverage [5], [6]. For web applications task becomes way more complicated and existing approaches could not be applied as is. Business logic is divided into client-side and server-side code. These two pieces of code are completely different, are implemented using different programming languages, frameworks etc. Communication is done using only HTTP requests. Current research aims to overcome this issues and to propose a method that would be able to adopt existing techniques to achieve high branch coverage for web applications. We analyze application JavaScript code to discover set of possible user actions as well as user inputs these actions depend on. JavaScript action handlers are analyzed and all dependencies of webpage elements and user inputs are extracted from source code and treated as function arguments. Then each possible action could be treated as a JavaScript function call with number of arguments.

If a JavaScript function makes AJAX calls to server-side code, then client-side and server-side code are treated as a combined function. Abstract syntax trees are constructed both for the client-side functions and for the server-side function. While it is not possible to automatically convert code from one language to another, it is possible to construct ASTs for JavaScript handler and callback, as well as for the server-side code, and to replace AJAX request node with a subtree that describes the server-side source code. Resulting AST would describe client-side code together with server-side code and could be represented as one function with a set of arguments. Given a function that emulates behavior of the web application problem of the test generation could be treated as a reachability problem and exiting tools to discover test data could be applied.

The rest of this paper is organized as follows. Section II contains a brief overview of the existing approaches and tools for web applications testing. In Section III the proposed approach is presented. Detailed AST generation and post-processing is given in Section IV. Section V tells about

developed and utilized tools that automate described approach. An example of the code where presented approach could be useful is presented in Section VI. Section VII describes current limitations of the approach and gives an overview of the future research plan.

II. Related Work

There are number of approaches and tools aimed at testing web applications, but none of them is able to automatically generate tests cases that would test the web application as a whole. Existing tools are designed to test some part of the application but not the web application as a whole, which consists of an GUI, client-side JavaScript, server-side code and some tricky interaction between these parts. In the proposed approach we make an attempt to combine different existing tools functionality and to create a framework that would be able to test web application as a whole and to generate test cases automatically. Current section describes existing tools, state their pros and cons and briefly explain how some of these tools would be utilized in the proposed approach.

Web application is usually an event driven system where user generates events. Selenium[1] and Sahi[2] tools could be used as a driver that provides facilities to emulate user interaction with a web page, that include filling out forms with user input and mouse clicks. These tools can be used to play given sequence of actions, but not to generate these sequences. Manual generation of these sequences could be useful for regression testing but it is impossible to achieve high code coverage using them manually. In proposed approach we generated these sequences automatically and utilize Selenium tool to execute generated test cases.

In [8] a framework for automated testing of JavaScript web applications is proposed. The goal of that research is to develop scalable and effective algorithms for automated testing of JavaScript applications to discover common programming errors of various kinds, including uncaught exceptions, improper type coercions or misuses of the browsers built-in libraries, invalid HTML, AJAX HTTP errors, and violation of assertions. While described tool seems to be useful for analyzing source code of the client-side, which is implemented in JavaScript, it does not take into consideration server-side code of the web applications as we try to do in our approach. That would be a serious drawback for complex applications, as it is common to implement most of business logic of the application on the server side. Moreover behavior of client-side code can not be analyzed correctly as it depends on AJAX callbacks which result could be estimated only if server-side code is analyzed.

Other existing tools for testing JavaScript of the web application are mentioned in [8]. The Crawljax/Atusa [9] project uses dynamic analysis to construct a model of an applications state space, from which a set of test cases is generated. The Kudzu project [10] combines the use of random test generation to explore an application's event space with the use of symbolic execution for systematically exploring an applications value space (i.e., how the execution of control flow paths depends on input values). Kudzu relies on an elaborate model for reasoning about string values and string operations, which are frequently manipulated by JavaScript applications.

There are number of researches [5], [11], [3] that address problem of generating tests with high branch coverage for common applications code that does not have specifics like client-server interaction. In proposed approach we treat server side-code as a common application with a specified entry point and set of arguments. Existing tools and frameworks are utilized to generate set of input values to cover all possible code branches in web application source code.

III. Proposed Approach for Generation of Test Suites with High Branch Coverage

Our approach is to convert the test case generation problem into a reachability problem. We observe that a web application can be described by its DOM state and set of possible user actions and corresponding JavaScript callbacks. Therefore problem of testing web application as a whole is separated into two tasks:

1) Discover all possible states of the web application.
2) For each discovered state cover each possible action and callback with a test case.

In [4] we addressed problem of creating models of web applications and proposed an approach to discover all possible different web application states. In current research we address the second task. For a given DOM state of the web application set of tests need to be generated that would cover all feasible execution paths. To achieve this goal we require a model of the system that describes all possible execution paths considering both client- and server-side code. In the developed approach we build an AST that describes control and data flow of the web application as a whole.

Proposed approach consists of the following steps:

1) Analyze client-side source code of the web application and build an AST that describes all possible JavaScript code branches.
2) For each graph node that contains an AJAX call to the server find corresponding server-side function. Source code of the server-side function is analyzed and also an AST is built.
3) Each node with AJAX call is replaced with a corresponding constructed server-side AST.
4) Final AST that describes the system in whole is post-processed to discover set of user inputs that conditional branches depend on.
5) AST is then represented as a C# or Java function that has user inputs as arguments and all condition branches are presented in its body. From this point of view problem of web application testing becomes problem of testing a common function with a set of arguments. Existing technologies and tools are utilized to generate test data that provide high branch coverage for the given function.
6) For each feasible execution path a Selenium [1] test case is generated where discovered test data is used as a user

input that would make the web application cover the desired branches.

7) Application specific business logic test oracles could be optionally added to the generated test cases.

Developed approach makes it possible to automate generation of a test suite with high branch coverage for a given web application state. Combined with earlier proposed state discovery algorithm [4] it is possible to generate test cases that cover web application source code in whole. Generated test suite would be able to discover common programming errors, including unhandled exceptions, runtime errors, undefined variable dereferences, invalid function calls, violation of assertions, invalid DOM operations and etc. Application specific business logic test oracles is impossible to generate automatically and need to be added by developers in the form of JavaScript or server-side assertions that would be also checked automatically during test execution.

Detailed description of the approach steps, problems that were solved and developed tools are presented in the further sections.

IV. Constructing a Simplified AST that Describes a Web Application as a whole

To cover web application's source code with tests a representation is required that describes both client-side code and server-side code in a unified form. In general, there is no existing technique to convert source code from one programming language into another. Moreover, while client-side business logic is always implemented in JavaScript, there are wide variety of programming languages and frameworks that could be used for server-side implementation, which makes task of converting all the source code to one language even more infeasible. But for the purpose of testing higher level of abstraction could be enough, so both server and client-side source code could be converted to some common representation that would be enough for the given task. We propose to use an Abstract Syntax Tree representation for this task.

Basically to cover all possible code branches with test cases a Control Flow Graph of the application should be analyzed. In a control flow graph each node in the graph represents a basic block, i.e. a straight-line piece of code without any jumps or jump targets; jump targets start a block, and jumps end a block [13]. But in practice such CFG representation would not be enough to generate test data automatically because it lacks data flow information. Condition in the flow graph nodes depend both on global and local variables. Such variables could be arguments to the function (like user inputs that are not modified during the execution), but also variables could be introduced locally that stand for loop counters, post-processed argument values and etc. Basic blocks of CFG could contain statements that modify such variables and excluding such information from the model would make analysis impossible.

Another way to describe source code using higher level of abstraction is Abstract Syntax Tree. AST is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is 'abstract' in the sense that it does not represent every detail that appears in the real syntax [17]. While CFG does not contain all the required information for the analysis, AST often contains superfluous information. Code statements that do not affect data flow of control flow of the program should be left out of the scope of the analysis.

In our approach we build a simplified AST that describes the application behavior. An algorithm is developed to construct an AST that describes web application page in whole:

1) Gather all the JavaScript source code in one place. Normally a web page's JavaScript source code could be divided into different .js files, that are included in the header block, as well as located in different parts of the HTML file. All occurrences of JavaScript code are discovered by searching for the pattern: `<script src=""></script>`
2) Build AST for the given JavaScript code. In practice it would be set of ASTs, as some parts of code will not be connected to other. For each separate piece of code an AST is built.
3) For each AST find an event that triggers corresponding code. JavaScript code is mostly event driven. Part of the code is executed on page load, but, usually, most of the code is located in action handlers and in callbacks for server calls. For each root node in AST a trigger is defined: page event, like timer or on load, user action or a server callback.
4) Locate all nodes that represent AJAX server calls. Server url information can be extracted from this node, as well as callback function. While in some cases server side function could be determined automatically, there are cases when automatic detection by given url is impossible (complex nginx configs, url mapping, redirects and etc.) and developer should provide such mapping manually to assist test data generation. An AST is built for server-side code and corresponding node in JavaScript AST is replaced by the constructed AST.
5) While arguments list passed from JavaScript directly maps to server function arguments, server-side arguments' names may differ from client-side code names. To preserve correct data flow server-side ASTs is post-processed and all names are replaced with its corresponding JavaScript names.
6) Each return node in the server-side AST is replaced with the corresponding callback AST. If the server-side call returns a value then function's result is introduced as a local variable and a corresponding assignment operator is added.
7) DOM access operators that used to fetch user input are replaced with function arguments. `"replacedN"` is introduced as a function argument and DOM access is replaced with variable access. For further generation of Selenium test sequences an object is created which

maps introduced variables to the actual objects in the following form:

```
{"replaced1":"document.getElementById(
'myTextInput').value()",
...,
"replacedN":"$('.ageInput').val()"}
```

When generating Selenium test cases arguments to the examined function are passed by filling corresponding form elements with discovered by the constraint solver values.

8) All superfluous nodes are removed. If a node does not affect data flow or control flow of the application then it is unimportant for the test data generation process and could be removed. JavaScript code normally contain a lot of DOM manipulation operators that are used to update page correspondingly to the new state. All these operators are useless on the desired level of abstraction and therefore are removed.

A set of ASTs is built after the completion of the listed steps. Constructed set describes web application page in whole, its data and control flow both for client-side and server-side functions. Having such description of the web page problem of test data generation can be converted into reachability problem. Each path in the AST is described as a list of constrains (conditions from the conditional branches) and a set of arguments that have to satisfy these constrains for the execution to follow the selected path.

List of constraints and arguments need to be analyzed to discover unsupported items. Server-side code implements business logic and often contain external dependencies like:

- database resources;
- file system;
- session variables;
- external web services calls.

For paths in the AST that contain any of the listed dependencies automatic test cases could not be generated, as tests would not be able to emulate state of the external objects. Only paths that do not have such dependencies would be a valid input for the constraint solver. Listed external dependencies are a strong limitation for any approach that attempts to automate testing, and ours is not an exception.

We propose to solve this problem by manually introducing mock objects that would emulate database, file system and other dependencies. Our tool could assist in creation of such objects by automatically creating list of required APIs for these mock objects. Developer would be provided with the generated class interfaces he would implement manually.

V. Tools to implement proposed approach

Goal of the proposed approach is to automate testing of web applications. Developed approach consists of number of different steps and for each step a special tool is required to automate the process. While for some steps existing tools could be utilized for other steps proof-of-concept tools were developed. A proof-of-concept framework that implements the proposed approach is being developed using Python 2.7 and JavaScript programming languages.

Static analysis of the web page is performed using a developed tool, the utilizes Selenium framework functions [1] and LXML python library [19] to parse web page and to build a DOM tree. Earlier developed tools [4] are utilized to discover and to compare web application states. Client-side JavaScript code is analyzed using Treehugger library [12], which is able to build an AST describing source code. Treehugger also supports usage of jQuery framework in JavaScript code that is often used to manipulate web page's DOM.

Proposed approach would suite for any language used for server-side development, the only requirement is that an AST with the correct syntax could be constructed for this language. Currently PHP and Python languages support were added during the research. ASTs for the PHP code are built using PHP-Parser [14]. Python source code AST representation could be retrieved using a built-it module [15].

Once all ASTs are brought to the common format merging and post-processing is performed by a developed python tool: superfluous nodes are removed, DOM inputs access are replaced by function arguments, AJAX calls and callbacks are replaced by the corresponding ASTs. All ASTs are stored in text files in the form of string representation.

In order to generate test data that would cover all execution branches we utilize concolic testing tools. Concolic testing is a hybrid software verification technique that interleaves concrete execution (testing on particular inputs) with symbolic execution, a classical technique that treats program variables as symbolic variables. Symbolic execution is used in conjunction with an automated theorem prover or constraint solver based on constraint logic programming to generate new concrete inputs (test cases) with the aim of maximizing code coverage [18]. There are list of tools that implement described technique for different programming languages: PathCrawler, PEX, DART, CUTE. We propose to use Microsoft Pex that is publicly available as a Microsoft Visual Studio 2010 Power Tool for the NET Framework [5]. PEX tool supports C# source code therefore final AST set need to be converted to C# source code. A converter is being developed during the research to perform this conversion using a straightforward algorithm.

Once test data is generated test case for each data set are generated. Test cases are implemented in python language and utilize Selenium WebDriver, a collection of language specific bindings, to emulate user actions and inputs in the given browser. PyUnit tool can run the generated test suite automatically.

VI. Illustration of the proposed approach

Proposed approach could be illustrated with an analysis of a part of the website registration form. User inputs are often checked at server-side code for different business logic constrains. For example a valid value for height of a new user has to be more than 50cm and less than 250cm. Client-side code contains an `input` form element and for its `change` event a jQuery handler is introduced. Inside handler an AJAX

call to server is made and entered value is validated with the following PHP code:

```
if (v > = 50 and v <=250) {
return "true";
} else {
return "false";
}
```

JavaScript callback parses returned value and displays an error message if `"false"` was returned. For the given example a combined AST is generated:

```
Function( "handler",[FArg("v")],[ If( Op(
"&&", Op(">=",Var("v"),
Num("50")),Op("<=",Var("v"),Num("250")
)),Block(...),Block(...))])
```

PEX tool is used to find input values for two possible branches with the following constrains:

1) `v >= 50 && v <= 250`
2) `!(v >= 50) || !(v <= 250)`

Test data generator easily finds values that suit for both execution paths and knowing that `v` stands for `$("#height-input").val()` page element two Selenium tests that would trigger two possible execution paths are generated.

VII. Conclusion

Web applications are built using client-server model and operate with callbacks rather than then sequential method calls. For these reasons tools and approaches developed for structured and object oriented programs can not be applied out-of-the box for web applications quality assurance. In this research we made an attempt to adapt existing test techniques to support web applications specifics. For each possible user action combined syntax tree is constructed to describe client-side and server-side source code together. Existing concolic testing tools are applied to the generated source code that behaves similar to web application. Discovered test data and Selenium WebDriver are used to generate a set of test cases with high branch coverage for the given application.

In the further research it is planned to develop a strategy that would assist developers in creating mock objects to make it possible to test functionality that depend on external objects like database and file system. Asynchronous nature of web pages interaction with server-side code also needs further investigation. Nevertheless proposed testing technique could be used to automate testing of the web applications and significantly improve software quality and defect detection rate.

References

[1] Antawan Holmes , Marc Kellogg, Automating Functional Tests Using Selenium, AGILE 2006: .270–275

[2] Web test automation tool. http://sahi.co.in/w/sahi

[3] Zakonov A., Stepanov O., Shalyto A.: GA-Based and Design by Contract Approach to Test Generation for EFSMs. IEEE EWDTS 2010: 152–155.

[4] Zakonov A., Shalyto A.: Automatic Extraction and Verification of State-Models for Web Applications. Lecture Notes in Electrical Engineering, 2012, Volume 133, Part 1, 157-160

[5] Tillmann N., Halleux J.: Pex – White Box Test Generation for .NET. Lecture Notes in Computer Science, 2008, Volume 4966/2008, 134-153,

[6] Pandita R., Xie T., Tillmann N., Halleux J.: Guided test generation for coverage criteria. ICSMIEEE Computer Society (2010) , p. 1-10.

[7] Bartak R., Salido M., Rossi F.: New Trends in Constraint Satisfaction, Planning, and Scheduling: A Survey. The Knowledge Engineering Review, 2004, Cambridge University Press

[8] Artzi, S., Dolby, J., Jensen, S.H., Moller, A., Tip, F.: A framework for automated testing of javascript web applications.In ICSE(2011)571-580

[9] A. Mesbah and A. van Deursen. Invariant-based automatic testing of AJAX user interfaces. In Proc. 31st Int. Conf. on Software Engineering, ICSE 09, May 2009.

[10] P. Saxena, D. Akhawe, S. Hanna, S. McCamant, D. Song, and F. Mao. A symbolic execution framework for JavaScript. In Proc. 31st IEEE Symp. on Security and Privacy, S&P 10, May 2010.

[11] Arcaini, P.; Gargantini, A.; Riccobene, E.: Optimizing the automatic test generation by SAT and SMT solving for Boolean expressions. ASE, 2011 26th IEEE/ACM.

[12] JavaScript AST library. https://github.com/ajaxorg/treehugger

[13] Control flow graph. Wikipedia. http://en.wikipedia.org/wiki/Control_flow_graph

[14] PHP-parser and AST builder. https://github.com/nikic/PHP-Parser

[15] Built-in module for AST in Python. http://docs.python.org/library/ast.html

[16] Python module offering solvers for Constraint Solving Problems. http://labix.org/python-constraint

[17] Abstract syntax tree. Wikipedia. http://en.wikipedia.org/wiki/Abstract_syntax_tree

[18] Concolic testing. Wikipedia. http://en.wikipedia.org/wiki/Concolic_testing

[19] Feature-rich and easy-to-use library for processing XML and HTML in the Python language. http://lxml.de/

MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors

Alexander Kamkin
Software Engineering Department
Institute for System Programming of RAS
Moscow, Russian Federation
Email: kamkin@ispras.ru

Andrey Tatarnikov[1]
Software Engineering Department
Institute for System Programming of RAS,
National Research University Higher School of Economics
Moscow, Russian Federation
Email: andrewt@ispras.ru

Abstract— **Test program generation plays a major role in functional verification of microprocessors. Due to tremendous growth in complexity of modern designs and rigid constraints on time to market, it becomes an increasingly difficult task. In spite of powerful test program generators available in the market, development of functional tests is still known to be the bottleneck of the microprocessor design cycle. The common problem is that it takes significant effort to reconfigure a test program generation tool for a new microprocessor design. The model-based approach applied in the state-of-the-art tools, like Genesys-Pro, still does not provide enough flexibility since creating a microprocessor model is difficult and requires special knowledge and skills. The article suggests an approach to ease generator customization by using architecture specifications that describe the microprocessor behavior at a higher level. The approach is aimed at facilitating development of architecture models and, thus, minimizing time required to create functional tests. At the moment, we are working to implement a new generation of the test program generator MicroTESK that can be easily configured for various microprocessor architectures.**

Keywords—microprocessor design; architecture description languages; test program generation; functional verification; model-based testing.

I. INTRODUCTION

As modern microprocessors are becoming more and more complex, *functional verification* is becoming an increasingly difficult task. It is typical that up to half of resources spent on microprocessor design is devoted to verification. The most common approach to verification of microprocessors at a core level is *test program generation* (*TPG*) [1]. *Test programs* (*TPs*) are instruction sequences that trigger device events and optionally check validity of the resulting state of the microprocessor. A tool that creates test programs for a given microprocessor architecture in an automated way is usually referred to as a *test program generator* or a *TPG tool*. A well-known problem of TPG tools is that test generation logic is often tightly coupled with the architecture-specific knowledge, which makes the tool hard to maintain. In fact, a frequent solution to handle new microprocessor architecture is to rewrite the existing generator from scratch. As we can imagine, it increases cost of the microprocessor development and causes significant delays in the delivery schedule.

To make a TPG tool more flexible, the architecture-specific part has to be isolated from the test generation core. That is usually called *model-based TPG* [1]. Platform-dependent knowledge includes mainly an *instruction set model* (*ISM*) and *testing knowledge* (*TK*), a collection of design-specific *test situations* (conditions to be covered by tests). Typically, scenarios for microprocessor verification are described manually in the form of *test templates* (*TTs*). In abstracto, the idea of the method can be expressed by the formula ***TPs = TTs + TK + ISM*** (TPs are generated on the base of TTs, which are described in terms of the ISM and TK). The model-based TPG is a time-proved approach having been implemented in the industrial tools, like Genesys-Pro [1] and RAVEN [2]. However, creating a microprocessor's ISM and TK is rather difficult and requires special skills that verification engineers are usually lacking for.

In this article, we present a concept of *reconfigurable TPG* and its implementation in the MicroTESK tool. The key idea is to use *architecture description languages* (*ADLs*), which are commonly used in the area of functional simulation [3], for TPG configuration. *Architecture specification* (*AS*) in ADL is used by the tool to automatically build the microprocessor's ISM and TK. In addition to ASs, MicroTESK utilizes light-weight *configuration files* (*CFs*) for some microprocessor subsystems. This is due to the fact that some elements (e.g., cache memory, address translation mechanisms and others) are difficult to describe in general-purpose ADLs. Usage of high-level specifications and automated extraction of ISM and TK make it easy to adapt the tool for new architectures and to reconfigure it for several revisions of the same design. One more important feature of MicroTESK is its ability to automatically generate TTs of certain types. Thus, to generate TPs, one needs only AS and CFs (***TPs = AS + CFs***).

The MicroTESK generation core comprises tools and libraries that allow working with different configurations in a uniform way. To accomplish this, a flexible tool architecture has been proposed. It is based on a rather general microprocessor meta-model, which makes it possible to all architecture-dependent components (result of translation of the AS and CFs) to be accessed via architecture-independent APIs.

The rest of the paper contains an overview of the existing approaches to TPG and describes the MicroTESK principles and architecture.

[1]This work is partially supported by RFBR 11-07-12075-ofi-m.

II. Related work and motivation

Hardware verification has always been a major issue for the research community. Over the last decades, a lot of hardware verification methods and tools have emerged. In fact, the idea of reconfigurable TPG is not new. It is based on a combination of well-known techniques. In this section, we will discuss the most significant of the existing approaches and industrial tools such as Genesys-Pro (IBM Research Lab) [1] and RAVEN (Obsidian Software Inc., now acquired by ARM) [2] implementing them.

Genesys-Pro is the best known TPG tool. It follows the model-based approach and operates with two kinds of knowledge: architectural model (ISM and TK) and TTs. To create an architecture model, some high level building blocks are provided. TK serves as a basis for creating TTs that describe verification scenarios. In TTs, it is possible to define constraints on individual scenario instructions (e.g., boundary conditions, exceptions, cache hits/misses, etc.). For each instruction of the TT the tool formulates a *constraint satisfaction problem* (*CSP*) and generates test data by solving the CSP. A known disadvantage of Genesys-Pro is that it is difficult to model instructions affecting memory devices [4]. Therefore, there are reasons to think that Genesys-Pro is hardly reconfigurable if significant modifications of the memory devices are required.

Another popular industrial solution is RAVEN. It is a tool that can generate fully random, semi-random or user-directed TPs for microprocessors. The tools components are separated into architectural models and TTs. Architectural models are developed by the tool vendor in collaboration with microprocessor manufacturers. For custom designs the *generator construction set* (*GCS*), a C++ API to the RAVEN core, is provided. There is no detailed information available on this technology. However, creating a model for RAVEN is unlikely to be an easy task for a verification engineer. In our opinion, it implies close interaction with the tool's developers, which is not convenient and will inevitably cause delays in the microprocessor verification process.

An interesting ADL-based approach to automated TPG is discussed in the work of Prabhat Mishra and Nikil Dutt [5]. It presents a concept of *graph-based functional test generation*. The approach uses the EXRESSION ADL to build a graph-based coverage model. The extracted model is automatically processed to extract test situations that will be covered by generated TPs. The test generation procedure is based on *model checking* (test is constructed as a *counterexample* for the negation of the target test situation). Heon-Mo Koo and Prabhat Mishra in their work [6] discuss a TPG technique that uses SAT-based *bounded model checking* (*BMC*) to generate TPs. Such an approach gives better results in terms of time and space required for counterexample generation compared to ordinary model checking that suffers from the state explosion.

Finally, it should be said that Institute for System Programming of RAS (ISPRAS) has already done some research and development on the TPG topic [4][7]. The present article summarizes the ideas that have been accumulated in ISPRAS and provides an overview of the research project our team is working on at the moment. The main motivation of the work is to propose a convenient way to describe microprocessor architectures that would reduce effort needed to create ISMs and TK.

III. MicroTESK overview

MicroTESK is a reconfigurable TPG tool that uses ADL descriptions together with high-level configuration information to represent architecture-specific parts of the generator. By *reconfigurability* we mean an ability to easily switch to a new microprocessor design without having to modify the internal logic of the tool. General structure of MicroTESK is displayed in Figure 1. The tool uses two main types of input data: (1) design description and optionally (2) user-defined TTs. The former provides information about the target microprocessor, while the latter specifies scenarios to be reproduced in TPs. Outputs are TPs in an assembler language. MicroTESK functions can be divided into three major groups: (1) translating a design description into the ISM and extracting TK, (2) creating TTs on the base of the TK and (3) generating TPs from the TTs. In other words, to generate tests for the target microprocessor, we need to go through the following stages:

- Creation of a design description in ADL and configuration of the design subsystems. This task is performed by a *modeling engineer* who possesses knowledge about the microprocessor architecture.
- Translation of a design description and configuration into the architectural model (ISM and TK). This is done by a *translator* that uses unified building blocks from the *model library* to generate a model.
- Creation of TTs. TTs are created basing on the ISM and TK extracted from the design description. TTs can be either created automatically by a *TT generator* or provided by a *verification engineer*. The advantage of TTs is that they provide a flexible way to specify instruction sequences and instruction parameters that can vary depending on some conditions.
- Generation of TPs. TPs are generated by processing TTs. During this stage all CSPs formulated for test situations are solved and all instructions have their final parameter values assigned. In the end, a *TP generator* produces an assembler program which is compiled by a verification engineer into binary code and executed in a simulator or on a chip.

As we can notice, at each stage the tool works with data produced by the previous stage. This reduces dependencies between different components of the tool and facilitates their customization. For example, adding support for a new ADL will affect only the translator as the architectural model representation is independent of a particular ADL.

The next sections of the article describe the MicroTESK components in more detail.

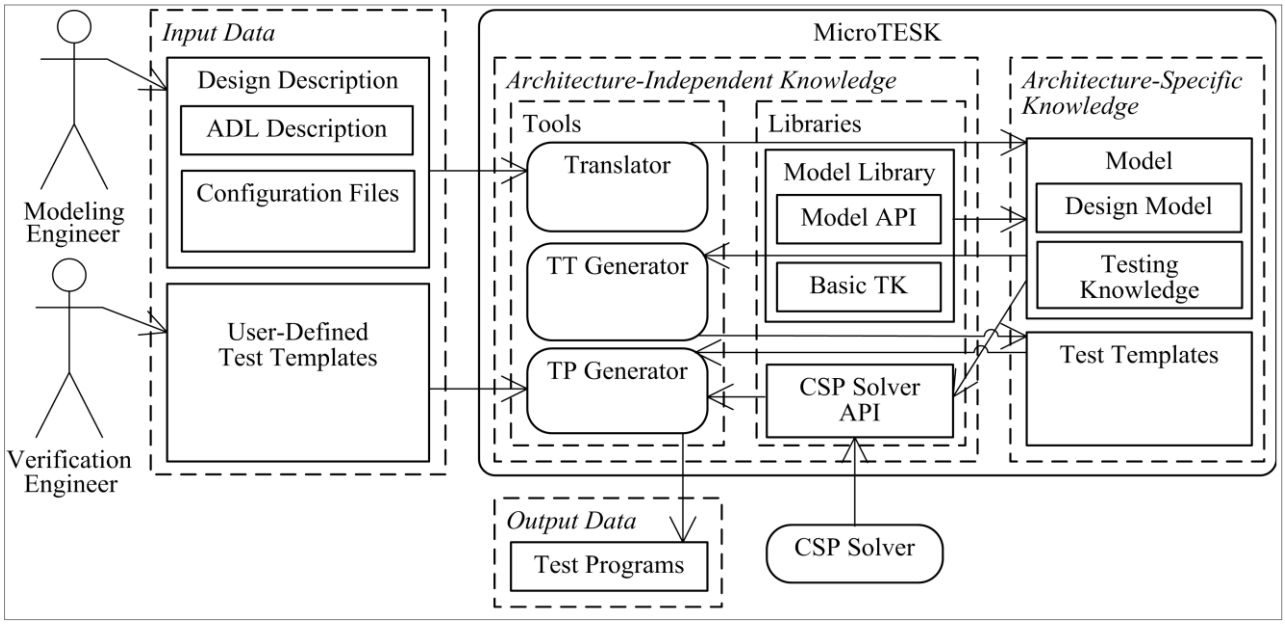


Figure 1. General structure of the MicroTESK TPG tool

IV. ARCHITECTURE MODELING

As it has already been said, MicroTESK makes use of ADLs to specify the target design architecture. At the moment, supported ADLs include nML [8] and Sim-nML [9]. nML is a formalism that describes a microprocessor at the instruction set level hiding unnecessary low-level details. The language is flexible and easy to use. Thereby, modeling a microprocessor architecture does not require significant effort. For example, a description of the integer addition instruction (ADD) from the MIPS instruction set architecture [10] looks as follows [4]:

```
op ADD(rd: GPR, rs: GPR, rt: GPR)
action = {
  if(NotWordValue(rs) || NotWordValue(rt)
  then
    UNPREDICTABLE();
  endif;
  tmp_word = rs<31..31>::rs<31..0> +
    rs<31..31>::rt<31..0>;
  if(tmp_word<32..32> != tmp_word<31..31>
  then
    SignalException("IntegerOverflow");
  else
    rd = sign_extend(tmp_word<31..0>);
  endif;
}

syntax = format("add %s, %s, %s",
  rd.syntax, rs.syntax, rt.syntax)

op ALU = ADD | SUB | …
```

As we can see, this notation is quite similar to the instruction's specification in the MIPS manual, which is shown below.

```
if NotWordValue(GPR[rs]) or NotWordValue(GPR[rt]) then
  UNPREDICTABLE
endif
temp ← (GPR[rs]31‖GPR[rs]31..0) + (GPR[rt]31‖GPR[rt]31..0)
if temp32 ≠ temp31 then
  SignalException(IntegerOverflow)
else
  GPR[rd] ← sign_extend(temp31..0)
endif
```

Basing on such specifications the microprocessor's TK can be automatically extracted. For example, analyzing the instruction description, we can derive three conditions that require attention:

1) The rt and rs general-purpose registers (GPRs) should contain sign-extended 32-bit values (bits 63..31 should be equal). Otherwise, the result of the instruction is UNPREDICTABLE. This means that under such a condition the microprocessor behavior is undefined and cannot be checked. Such situations should be avoided in TPs.

2) If the addition results in 32-bit two's complement arithmetic overflow, the destination register rd should not be modified and the IntegerOverflow exception should occur. Such a situation can be specified in a TT. When a TP is being generated the constraint solver engine will calculate exact values of the rt and rs GPRs to satisfy the constraint.

3) If the addition executes normally (does not cause an overflow), the sign-extended 32-bit result should be placed into the rd GPR. Such a condition can as well be used in TTs (for example, to be sure that some instruction does not raise exceptions).

ADL descriptions are used to build coverage models for individual instructions and to determine basic inter-instruction dependencies. It should be emphasized that models are composed from the building blocks provided by the model

library and independent of a particular ADL. Therefore, it is possible to use any ADL that provides enough information regarding the structure and behavior of microprocessors.

In addition to an instruction-level ADL description of the microprocessor, there is a need to specify some microprocessor subsystems, like memory management unit (MMU) and pipeline control unit (PCU), in more details. ADLs like Sim-nML are not suitable for describing these elements. At the same time, they should not be overlooked as it is very important to verify how a microprocessor handles events related to memory management and pipeline control. To provide specifications of these properties, special *configuration files* (*CFs*) are used.

MMU provides memory access protections, virtual-to-physical address translation and caching of instructions and data. It works with the main memory, cache memory (L1 and L2) and translation look-aside buffers (TLBs) that are used to accelerate virtual-to-physical address translation by caching latest translations. A cache or a TLB is represented by a memory buffer. At a logical level, each buffer is described as an array of sets of lines that can be specified as structures comprising several bit vectors called fields. A line stores a copy of memory data that has been recently read or written. Data is accessed by its address. When a buffer contains a line with a specified address the situation is called a hit; if it does not the situation is called a miss. When a miss occurs, the line is replaced with data stored in main memory at the given address. So, buffer configuration information includes the following attributes: set size (associativity), number of sets, line field description, address-to-index translation rule, rule for checking if a line and an address match and data displacement policy. To specify this information, we use CFs of the following kind [4]:

```
buffer L1 = {
   set = 4
   length = 128
   line = { tag:card(27), data:card(32) }
   index(addr:36) = { addr<8..2> }
   match(addr:36) = { addr<35..9> == tag<0..26> }
   policy = LRU
}
```

For the purpose of functional verification, there are two main situations that interest us: when a hit occurs and when a miss occurs. Both situations can be formulated as CSPs over the address being used to access data and state of the buffer [11][12].

Another important aspect related to architectural models is dependencies between instructions. Instructions change the state of the microprocessor and, thus, affect the behavior of subsequent instructions. For example, a precondition for the hit event is that corresponding data are loaded into cache, which is done by a previous instruction that accesses the same data line. To produce a complex instruction sequence that will give predictable results, we need first to simulate its execution to determine final parameter values of the dependent instructions. This is done at the final stage when MicroTESK processes TTs to produce TPs. The tool keeps a track of all events that occur in the model and provides this information to the TP generator Knowledge about possible dependencies between instructions is a part of TK extracted from the CFs.

V. CONSTRAINT SOLVER ENGINE

Important part of MicroTESK is a CSP solver engine. It facilitates generating test data and helps to achieve a better test coverage. Architecture specifications do not usually specify precise parameter values that lead to particular situations, but rather specify a class of possible values expressed as a set of conditions. For example, when we want to create a test for an integer overflow exception in the ADD instruction, we do not know values of parameter that cause the exception (in fact, there may be thousands of possible values). However, we know what conditions the resulting value should satisfy to recreate the situation. To generate parameter values that will make a test situation occur, the tool formulates a CSP and solves it with the help of the solver engine. The engine returns parameter values satisfying the constraint. Such an approach allows generating new test data from each time a TP is generated from the TT, which improves test coverage.

MicroTESK uses the SMT-LIB language [13] to formulate CSPs for test situations. CSP is expressed as a set of assertions that specify assumptions about values of input variables and results of operations performed with them. Modern solvers support bit vectors, which facilitates specifying constraints for data buffers used in different parts of microprocessor models (registers, cache, main memory, etc.). Below, there is an example of a CSP that specifies conditions leading to an integer overflow exception in the ADD instruction.

```
(define-sort Int_t () (_ BitVec 64))

(define-fun INT_ZERO () Int_t (_ bv0 64))
(define-fun INT_BASE_SIZE () Int_t (_ bv32 64))

(define-fun INT_SIGN_MASK () Int_t
   (bvshl (bvnot INT_ZERO) INT_BASE_SIZE))

(define-fun IsValidPos ((x!1 Int_t)) Bool
   (ite (= (bvand x!1 INT_SIGN_MASK) INT_ZERO) true false))

(define-fun IsValidNeg ((x!1 Int_t)) Bool
   (ite (= (bvand x!1 INT_SIGN_MASK) INT_SIGN_MASK) true
false))

(define-fun IsValidSignedInt ((x!1 Int_t)) Bool
   (ite (or (IsValidPos x!1) (IsValidNeg x!1)) true false))

(declare-const rs Int_t)
(declare-const rt Int_t)

; rt and rs must contain valid sign-extended
; 32-bit values (bits 63..31 equal)
(assert (IsValidSignedInt rs))
(assert (IsValidSignedInt rt))

; the condition for an overflow: the summation
; result is not a valid sign-extended 32-bit value
(assert (not (IsValidSignedInt (bvadd rs rt))))

; just in case: rs and rt are not equal
; (to make the results more interesting)
(assert (not (= rs rt)))
(check-sat)

(echo "Values that lead to an overflow:")
(get-value (rs rt))
```

MicroTESK provides a possibility to generate CSPs automatically on the base of the TK extracted from the design description. It should be said that TK's constraints are stored in a format that is independent from a particular solver. The current version of the tool uses the Z3 solver by Microsoft Research [14]. The TP generator interacts with the solver via solver-independent *CSP solver API*. This allows the tool to use different solvers.

VI. TEST TEMPLATES

TTs are an important part of the MicroTESK solution. The tool provides facilities to create and modify TTs by hand. Despite the fact that some amount of TTs can be generated automatically based on TK, to cover all possible situations, it is often necessary to create TTs manually or customize automatically generated ones. Therefore, an expressive and easy-to-understand language is needed. Generally speaking, a TT describes a class of TPs that verify microprocessor behavior in particular test situations. Whereas TPs represent sequences of commands in a processor-specific assembler, TTs provide a way to describe a test scenario at a more abstract level. Such an approach gives a lot of advantages in terms of flexibility. For example, it allows generating tests taking into account dependencies between related instructions, create tests for a whole class of similar instructions and specify test parameters as ranges of possible values or as random values instead of hard-coding them. It also helps organize groups of separate test scenarios in more complex test cases and set up parallel test execution.

To describe TTs, a special test template description language (TTDL) is used. In the current version of the tool, it is based on the Ruby scripting language, which is extended with special automatically generated libraries that provide all hardware-related features and perform interaction with the design model. Generally, the TTDL features can be divided into the following groups according to their purpose:

A. Achitecture-related statements

Include constructs to simulate generalized processor-specific assembler instructions and a list of supported registers. Both instructions and registers can be combined into families. The TTDL allows specifying a family instead of a precise element or an address range instead of a specific address. Thus, it is possible to vary the level of randomness in the generated tests from completely random to completely directed. Also, we can specify dependencies between instructions. For example, we can make them use the same registers or the same address, which is selected at random when being accessed for the first time.

B. CSP-related statements

Constraints can be applied to instructions to recreate test situations. Typically, constraint conditions are extracted from ADL specifications. For example, it can be a condition that causes an integer overflow exception. Constraints are stored in a special catalog of constraints that includes information about instructions (or classes of instructions) they can be associated with. Constraints can be extracted from a design specification automatically, created manually or provided with the tools as independent general TK which is common for different microprocessors.

C. Generation flow statements

Provide control over instruction generation sequencing. Sometimes the sequence of instructions in a TP may need to be varied depending on some conditions or even to be randomized to achieve a better level of coverage. There are several possible ways to specify how a TP can be generated:

- As a sequence of ordered instructions (the order is specified in the TT);
- As a sequences of specified instructions given in a random order;
- As a sequence of instructions some of which are repeated depending on some conditions;
- As a sequence of instructions that contain instructions (or subsequences of instructions) randomly selected from the specified set of instructions;
- As a set of instruction sequences that should be executed concurrently;
- etc.

The TTDL language provides language constructs that offer such facilities. The set of offered features can be extended.

D. Standard language constructs

Include constructs derived from the underlying scripting language such as control flow operators, variables, constants and assertion statements. Such constructs are necessary to describe complex scenarios, to specify shared instruction parameters, to use common constants and to add validity checks to test scenarios.

E. Infrastructure-related statements

Provide a framework for creating TTs. Include base classes and global objects needed to organize the structure of TTs and provide communication with design models and CSP solvers during test generation. Some features are architecture-specific and are generated from models automatically.

The TTDL provides facilities that suffice for most verification tasks. To simplify the test design process, MicroTESK provides an ability to automatically generate some types of TTs. This includes templates for single instruction tests (that cover all possible execution paths for all supported instructions), combinatorial tests (that generate short sequences represented by combinations of specified instructions) and random tests (that produce random instruction sequences). Automated generation of TTs is performed based on TK extracted from the design description. To generate more specific TTs, the design model can be extended with additional information about test situations and instruction dependencies.

To illustrate the use of the TTDL, an example of a TT for a MIPS microprocessor is provided below.

```
class MyTemplate < Template
  def test()
    data = [ [0xEF, 0xFF], [0x1EF, 0x1FF], [0xFEF, 0xFFF] ];
    data.each { |d|
      xor r0, r0, r0;
      ori r(2), r0, d[0];
      ori r(4), r0, d[1];
```

```
    ld tmp1=r(1), 0x0, r(2);; hit([L1(), L2()], [25, 50, 75]);
    ld tmp2=r(3), 0x0, r(4);; hit([L1(), L2()], [25, 50, 75]);
    dadd r(5), tmp1, tmp2;; overflow;
  }
  end
end # class MyTemplate
```

This TT represents a scenario that generates a set of instruction sequences parameterized with data stored in the array. The generated sequences load data located at the addresses stored in the data array. For the ld instructions, the TT specifies constraints related to cache events:

```
hit([L1(), L2()], [25, 50, 75]);
```

This statement means that the line that accesses a memory device should cause a cache hit event to occur. It specifies a set of target caches and probabilities of the hit event occurrence. For this line, MicroTESK will generate a list of possible combinations and will add an instruction for each of them to the resulting TP. Another constraint is used to make the dadd instruction generate an overflow exception. The TTDL provides a wide range of facilities to express test situations that involve complicated series of events.

VII. Conclusion

Verification of modern microprocessors requires a lot of effort and efficient instruments. An ability to quickly reconfigure a TPG tool for a new design is a crucial requirement. In this paper, we offered a solution to the problem. The paper contributes the following approaches: (1) using high-level ADLs and CFs to specify the configuration of a target design and (2) building TK from high-level specifications basing on behavioral characteristics of the target design and (3) automated generation of TTs and TPs based on TK. The approaches are applied in MicroTESK, the instrument our team is working on. It makes use of the nML/Sim-nML ADL to describe target microprocessor designs. This formalism uses a format similar to the notation used in microprocessor manuals, which significantly facilitates creating configuration description of target devices. Another important application of ADLs is that they serve as source of behavioral characteristics of a microprocessor. MicroTESK is able to extract TK from ADL specifications and use it as a basis for creating test scenarios. This simplifies the job of a verification engineer who being armed with this knowledge can start creating tests as a soon as MicroTESK has processed an ADL specification. TTs are another major feature of MicroTESK. It provides a flexible way to specify complex test scenarios. Test situations can be formulated as CSPs, which eliminates the necessity to provide exact values of instruction parameters to make a particular event to occur.

The architecture of MicroTESK facilitates customization. Designs models are created based on an API provided by the model library. They are independent of a particular ADL and can be processed in a uniform way. Also, MicroTESK includes built-in TK about situations that are common for different microprocessors. The template generation logic combines built-in TK and TK extracted from the architecture model to generate test scenarios, which allows automating the process of creating tests for basic test situations. The tool can be extended to support new ADLs and new ways to describe TK and TTs. As we can see, MicroTESK automates most of activities required to create tests for a target microprocessor design, which helps significantly decrease delays in the delivery schedule.

At the present stage of our research, we implemented a prototype that supports only a small set of the described features. The first version of the prototype was tried with several industrial microprocessors and their subsystems. The experimental results are provided in the work of Kamkin, Kornykhin and Vorobyev [4]. Our current plans are to develop a full featured product that could be used by microprocessor vendors. A further direction of research is to more extensively automate creation of TTs. This will require using more complex models and test generation techniques. To keep in pace with temps of growth in complexity of modern microprocessor designs, TPG tools should provide more facilities to automate test design.

References

[1] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov and A. Ziv, Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification, IEEE Design & Test of Computers, 2004, pp. 84-93.

[2] http://www.obsidiansoft.com/pdf/Datasheet.pdf

[3] P. Mishra, A. Shrivastava and N. Dutt, Architecture Description Language (ADL)-Driven Software Toolkit Generation for Architectural Exploration of Programmable SOCs, ACM Transactions on Design Automation of Electronic Systems, Vol. 11, No. 3, July 2006, Pages 626–658.

[4] A. Kamkin, E. Kornykhin and D. Vorobyev, Reconfigurable Model-Based Test Program Generator for Microprocessors, A-MOST, Berlin, Germany, 2011.

[5] P. Mishra and N. Dutt, Graph-Based Functional Test Program Generation for Pipelined Processors, In Design Automation and Test in Europe (DATE), Paris, France, pages 182–187, February 16-20, 2004.

[6] H. Koo and P. Mishra, Functional Test Generation using SAT-based Bounded Model Checking, CISE Technical Report 05-008, Department of Computer and Information Science and Engineering, University of Florida, 2005.

[7] A. Kamkin. Test Program Generation for Microprocessors, Institute for System Programming of RAS, Volume 14, part 2, 2008, pp. 23–63 (in Russian).

[8] M. Freericks, The nML Machine Description Formalism, Techical Report, TU Berlin, FB20, Bericht 1991/15.

[9] R. Moona, Processor Models For Retargetable Tools, Proceedings of IEEE Rapid Systems Prototyping 2000 June 2000, pp 34–39.

[10] MIPS64™ Architecture For Programmers, Volume II: The MIPS64™ Instruction Set, Document Number: MD00087, Revision 2.00, June 9, 2003.

[11] E. Kornykhin, SMT-Based Test Program Generation for Cache-Memory Testing, East-West Design & Test Symposium (EWDTS), 2009, pp. 124–127.

[12] E. Kornykhin, Generation of Test Data for Verification of Caching Mechanisms and Address Translation in Microprocessors, Programming and Computing Software, Volume 36 Issue 1, 2010, pp. 28-35.

[13] D. R. Cok, The SMT-LIBv2 Language and Tools: A Tutorial, GrammaTech, Inc., Version 1.1, 2011.

[14] L. Moura and N. Bjørner, Z3: An Efficient SMT Solver, Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2008, pp. 337–340.

Run-time monitoring for model-based testing of distributed systems

Vladimir Fedotov[1]
Institute for System Programming
Moscow, Russia
Email: vfl@ispras.ru

***Abstract*—Modern enterprise systems are highly distributed and heterogeneous. Apart from the latest attempts on leveraging distributed systems with SOA and SOA-like enterprise integration systems, testing still represents a major challenge.**

This paper discusses practical approach for managing the testing process for distributed systems based on transparent test environment, run-time monitoring of interactions within this environment and interaction model generation.

We also outline an approach for test case generation based on the interaction model and test coverage metric based on the coverage of interaction tree.

I. Introduction

Integration technologies are rapidly advancing since early 90-s, driven by consistently faster networking and better data storage. Being mostly a business domain, integration technologies constantly evolve, leaving in their wake various vendor-locked platforms as a legacy. This legacy forms a heterogeneous environment that is common for any enterprise company big enough.

Enterprise environment is often divided between several technology domains, each formed around a certain kind of solution like an integration broker, application server or service bus. Therefore environment as a whole is highly distributed. Bringing this environment together is a daily struggle for an enterprise IT.

The web-service stack of protocols is the latest attempt to deal with this issue. Centered on Web-service Definition Language (WSDL) web-service protocols provide standardized interface for integration components. Combined with stateless design it enables the strongest feature of web-services – compositions.

Composition is a web-service that acts as a client for several other web-services. Compositions can be stacked over each other to implement different tasks over existing functionality of the system, creating highly distributed environment. Enterprise systems often follow the pyramid pattern (Fig.1), wrapping enterprise applications with web-service interfaces and stacking several layers of compositions on top of them.

Interface complexity, the number of operations and operation parameters, grows from the top to the bottom of the pyramid. Low-level services may wrap an entire API of the underlying systems, requiring an expert knowledge of the business domain for client development. Top-level services implement very specific business processes and provide only basic interface that requires little knowledge of the system internals and can be exposed to the third-party developers.

Integration complexity, the number of outgoing requests for each incoming request, as defined in [1], grows from the bottom to the top of the pyramid. Low-level services are tightly coupled to the applications they wrap thus having zero integration complexity. Top-level services have multiple dependencies on the services below them that have dependencies of their own.

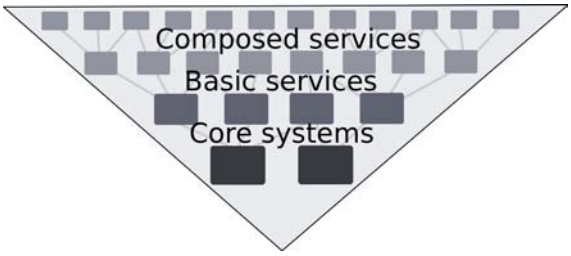


Fig. 1. Pyramid pattern in enterprise application integration

In this paper we would like to discuss the testing process for the systems described above. While web-services by themselves bring nothing new to the classic V-model testing process, compositions testing present an actual challenge, shifting focus from functional to the integration testing. Developing a methodology adjusted for testing of web-service compositions is a goal of the research described in this paper.

II. Motivation

Distributed heterogeneous nature of the enterprise systems presents several major issues that need to be dealt with in order to get meaningful consistent testing process. Other issues described below are the consequences of data-flow centered logic of integration components that makes them tightly coupled to data that is stored externally.

Typical enterprise system consists from several different technology platforms, that implement various, often proprietary, protocols. Applications supporting these protocols form a domain around the platform, which means that test environment is fragmented according to these domains as there are no connections between tests from different domains. Main

[1]This work is partially supported by RFBR 11-07-00084a, 11-07-12075-ofi-m grants.

consequence of environment fragmentation is lack of end-to-end tests that hampers a system testing stage of the process.

While individual components contain only integration logic i.e. transformation between different message formats, actual business logic resides in data stored in the core systems such as billing or resource management. Constraints that exist in this data describe what is possible and what is not in the system. Therefore testing of business logic requires knowledge about these constraints and a way to mine data corresponds to a certain set of constraints. Both of these tasks are extremely difficult as the data structures in the core systems may be incredibly complex. In practice constraint discovery and data mining often done in an informal way: by brainstorming the database structure or consulting with business experts.

The price of informality is that there are no guarantees of completeness for discovered constraint set. It may be too strict, so certain types of input data is not represented in test cases. Or it may be too loose, so certain test cases will fail for no apparent reason. The main consequence of this is that there is no appropriate way to measure test coverage. As test cases are data-driven, they should be executed with the same requests but different data, so typical coverage metrics, such as amount of covered web-service operations, become inadequate to actual test subject – business process implemented by the component composition.

III. Outline

Approach proposed in this paper can be logically divided into two parts: firstly we try to bring the test environment under control by developing a transparent test framework that offers us an ability to observe and control interactions within the test environment; secondly, we develop a technique for modeling these interactions, that gives us a way to formally reason about what is happening in the system.

Test framework described above is based on existing application integration solutions. We are extending an existing open-source enterprise service bus, that offers us various protocol adapters, message routing and transformation capabilities. By having centralized, possibly federated, test environment we deal with test environment fragmentation. Supporting the environment also becomes easier, as protocol adapters can be developed independently and do not affect other parts of the environment. Adapters are connected to the Normalized Message Router that transforms various message formats into canonical one, thus making end-to-end testing easier.

Another important feature of our framework is an interaction monitoring. It provides us with a bridge between real system and a model. As it seems impossible to derive interaction model from component specifications, we see run-time monitoring as a best possible alternative. Run-time monitoring at the unit-testing stage of process helps us to create behavior models for independent components. At the later stages interaction monitoring helps us to determine the outcome of the test executions and coverage reached.

The goal of model generation is to provide binding between data-driven test cases and composition behavior to deal with data constraints discovery issue. Model is built bottom-up, starting from request-reply pairs for individual component and growing to composition of models of several components. It is presumed that model generation starts at the unit testing stage, where data mock-ups are used. At this stage, model describes what types of behavior are possible for an individual component, later we bind them to an actual data, representing an equivalence class.

When interaction model for component composition is ready, we generate test scenarios that represent a certain interaction path within a system. Test stimulus is represented by a message that would be sent into composition entry point, while reaction is a set of all consequent interactions that happened within a system. Stimuli that have the same structure, but contain different data may produce an entirely different interaction pattern. The power of the approach is in testing various interaction patterns that may be hidden behind the entry point.

Essential feature of our approach is test coverage metric. Data complexity in an enterprise system often prevents instantiation of certain classes of data as we are unable to satisfy constraints that exist in the system. Well-managed testing process always aims for a perfect balance between risks and man-hours, so we need a tool to evaluate an amount of work required for test data instantiation and compare it to risks coming from not testing on this data. Our test coverage is measured as coverage of possible interaction patterns in the composition. This coverage metric includes not only coverage of reachable request-reply pairs, but also coverage of reachable data classes discovered on earlier stages of process.

IV. Approach

A. Transparent environment

Basis of our approach is the observable environment. While evaluating properties of the components under test we presume that interactions between these components are observable.

In practice it might be difficult to achieve such level of transparency. For example, there is no easy way, known to us, to observe database interactions. Also interactions over proprietary binary protocols, common for message queues, are observable, but not decodable. Still, most interactions are done over HTTP, with messages formatted in XML, so they are easily observable and readable.

Second part is to bring all the observations together in a single framework. Surely a bunch of HTTP sniffers here and there would not do much for our goal. As a solution we suggest the existing open source ESB platforms like ServiceMix, Synapse and others.

First of all, ESBs already have a lot of adapters for different protocols, thus widening our reach for many different platforms. Second, the concept of an ESB presumes existence of single point of observation for everything that happens inside a system. Third, messages passed into an ESB are normalized to canonical form, so we are relieved from a burden of handling different message formats.

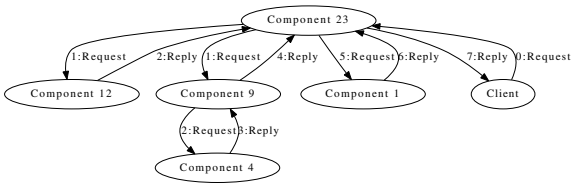


Fig. 2. Interaction graph

We have not yet reviewed all available ESB platforms, Apache Synapse looks most promising due to ease of its configuration by XML-based configuration language, but protocol support is somewhat limited in comparison to other platforms. We will also look into possibility of developing our own solution on top of the existing ones.

The role of the framework described above is to provide a new entity of the system - an observer. Observer should be able to log, analyze, transform and re-route messages passing through him. As we actually see interactions within the environment only if they are visible to observer, modeling the environment actually means modeling the observer's state. We see an observer's state as a queue of incoming messages. In a certain period of time message queue gets processed which basically means that messages get sent to their destinations. Queue-based processing gets us a handy abstraction of time for our model. Instead of dealing with continuous time, we have discrete time represented by a message polling interval of the observer that can be imagined like a turn in a turn-based computer game. For example, messages are considered concurrent if they were retrieved on the same turn and sequential if message B followed message A exactly on the next turn.

B. Interactions model

The first and easiest step of our approach is creation of a connection graph of the system, like the one shown on the Fig.2, that basically represents connections between components. This graph is created by monitoring of the message-flow on the observer and requires almost no processing other than message headers where message destination resides.

The second step is the creation of the interaction trees shown on the Fig.3 that maps the whole interaction between components to a single message that started it. To create an interaction tree we should implement basic rules for message correlation:

1) Messages are concurrent if they were sent on the same turn
2) Messages are sequential if the second message was sent on exactly next turn and destination of the first message matches to source of the second message
3) If there is no messages in the observer's queue, the interaction has ended

The third step is to discover relations between actual request data and component behavior. This step is necessary for a meaningful model as our components are data-driven and their behavior may depend not only on the type of incoming data, but also on its semantics. By developing this technique further we hope to implement a set of behavioral heuristics for discovering common dependencies in the input data, such as message ordering and equivalence classes.

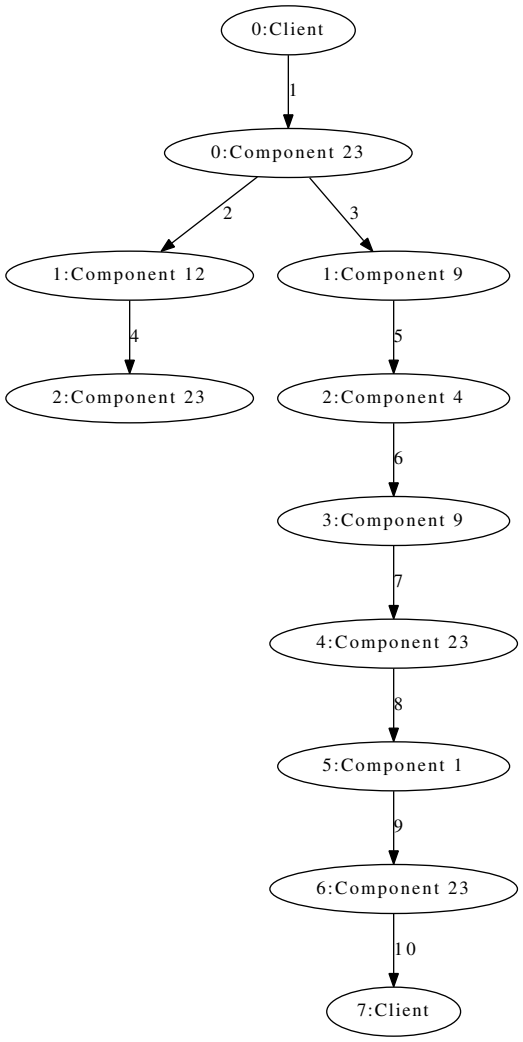


Fig. 3. Interaction tree

C. Test generation

Final step of our approach is a test generation based on the interaction model enriched with the discovered classes of input. To test a certain composition we need to derive the minimal set of test stimuli from a model that is able to cover all reachable branches of the interaction tree.

This task is achievable in case we would be able to discover and correlate inputs properly, the problem here is that these tests would be abstract i.e. they would contain a description of classes of data instead of a real data. Real data gathered by observer may become unusable in case of update operations, as the updated data no longer represents its original class.

Currently we see no possible general approach for discovery of concrete data instances, so we presume it is done manually. As it is certainly possible that some classes of data would not have concrete instances (because we were unable to find them,

not because they do not exist), it is vital to develop a coverage metric for these test sets.

D. Model semantics

Finally we would like to discuss a topic somewhat unrelated to the practical application of our approach, but essential for its further development – formal semantics of our model. Currently we are considering several different semantics. First one is widely used LTS semantics [2] [3] [4]. LTS model represents a system as a set of vertices - states and a set of labels - actions that perform a transition from one state to the other.

As we discussed earlier, we consider components as a stateless entities, so it may be unclear how to model them using LTS semantics. We propose a slightly different approach for using the same semantics. As we are modeling a data-flow in the system, we represent state as a message in an observer's queue and transition as an action performed by a component that results in transition to a new state i.e. getting new message in a queue. This approach is expressive enough, but there are certain difficulties in its practical application.

First of all, as we discussed earlier, our inputs have state of their own that should be included in the model. In reality it is some sort of the global state represented by a set of attributes stored in a database, but we think that modeling a global state is not exactly a good idea, because it brings unnecessary complexity to a model without any real value.

We deal with that issue by splitting our state, a message in reality, in two parts: implicit and explicit. Explicit part of the state is an actual message received, while implicit part is an associated context that is hidden from us somewhere in the system. By doing so we get rid of non-determinism we showed earlier, where two same messages may produce different reactions. Here it means that only explicit parts of the messages were equal, and implicit parts were actually different, so we have two different states and there is no indeterminism in these cases. Declaring the two states different is done solely by looking on the results of the same actions, so we do not need to actually compare the implicit parts.

Another option we are considering is somewhat less known actor semantics [5] [6]. Actors model was introduced by Hewitt in 1973 [7] and was supposed to model concurrent systems as a set of related entities – actors. Actors communicate via reliable messaging and have a state of their own. For every incoming message actor can create more actors, send more messages or change its own state.

Most recent and successful implementation of an actor model is done in the Scala language [8], which served for us as an inspiration to look into actor semantics. Scala provides an actor framework for implementation of concurrent systems in a clear and concise way that differs a lot from a traditional locking mechanism.

Model-checking for actors modeled in Rebeca language [9] was successfully implemented in Modere [10] model-checking engine. For now these are the only works in formal verification for actors, but while recent development of an actor model itself was not very active, we see its expressiveness in our domain as a big advantage. Components, especially web services, can be naturally described as actors. Message passing as a way of communication also fits naturally in our approach. Another big advantage is model scalability – a way of composing smaller models into larger ones described in [11].

V. Related work

Testing and verification of the distributed systems is a very popular field of academic research, mostly due to recent peak of SOA-related technologies. The formal specification language - WSDL and the composition description language - BPEL provided by web services attract attention as they seem to be very prominent tools for implementation of various model-based testing techniques.

The ideas that are closely related to ours and, more importantly, already implemented in the Plastic Validation Framework [12] were expressed in the works of Bertolino et al. [13] [14] [15]. The concept of the model-based generation of the test environment, expressed in [14] is very close to our approach. Unfortunate downside of the proposed methods is their limitation to the domain of web services. There also no clear description of the data binding mechanism as web services are described as stateful entities.

Another important work related to our topic, done by Sharygina and Kröning and included in [16], discusses the application of the model-checking techniques in the domain of web-services. Being a preliminary work it only discusses the model checking for certain properties such as deadlocks, but can be easily extended for more practical cases.

Castagna et al. [17] developed the theory of contracts for Web services. These contracts are used as behavioral descriptions of Web services and offer concise and formal way for reasoning about their properties.

Ferrara [18] developed the process algebra approach for reasoning about BPEL services. This approach maps formal abstracts to concrete web-service implementations done in BPEL4WS language.

Textor et al. [19] proposed formal workflow model for SOA monitoring. This model is used for Quality-of-Service monitoring for enterprise applications. Described approach was successfully implemented for self management of the actual enterprise system.

VI. Conclusion

In this paper we have introduced a model-based approach for testing distributed systems. Our approach has strong emphasis on practical application and is based on the run-time monitoring of the system. To enable such monitoring in real enterprise systems we develop transparent test environment that acts as an observer for interactions between components of the distributed system. We use existing open-source ESB as a technology platform for the test environment.

We have outlined an approach for generation of a model of distributed system that is based on observing behavior patterns

of the individual components. This model is composable and can be used throughout all stages of the testing process, from unit-testing up to end-to-end acceptance testing. The main feature of the model is that it allows binding component behavior to the semantics of the input data. Described model is used for generation of tests that cover possible interaction paths and input data classes.

We have also discussed suitable formal semantics for a described model. LTS semantics is common for model-based techniques, but cannot be applied in a straightforward manner because of the stateless design of the system components. Instead of modeling the component state, we model the state of the observer that is represented by a message or multiple messages for concurrent interactions. Another suitable semantics is the actor model, introduced by Hewitt and implemented in Erlang and Scala programming languages. Despite being unpopular for means of formal verification, we see the actors semantics superior to others mostly because of scalability of the models defined with actors.

Presented work is still very much in progress. Description of the model and test generation techniques is preliminary and would be improved in future. We also plan to put more efforts in researching the actors semantics.

REFERENCES

[1] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, no. 4, pp. 308–320, 1976.

[2] J. Tretmans, "Model Based Testing with Labelled Transition Systems."

[3] V. Kuliamin, "Component architecture of model-based testing environment," *Programming and Computer Software*, vol. 36, no. 5, pp. 289–305, 2010. [Online]. Available: http://dx.doi.org/10.1134/S036176881005004X

[4] I. Burdonov and A. Kosachev, "Conformance testing based on a state relation," *Proceedings of the Institute for System Programming of RAS*, vol. 18, pp. 183–220, 2010.

[5] C. Hewitt, "Actor Model of Computation: Scalable Robust Information Systems," pp. 1–29, 2011.

[6] S. Smith, I. A. Mason, and C. Talcott, "Towards a Theory of Actor Computation."

[7] C. Hewitt and P. Bishop, "A universal modular actor formalism for artificial intelligence," *Joint conference on Artificial intelligence*, pp. 235–245, 1973.

[8] M. Odersky and L. Spoon, "Programming in Scala," 2008.

[9] M. Sirjani and M. M. Jaghoori, "Ten Years of Analyzing Actors : Rebeca Experience."

[10] A. Movaghar, "Modere : The Model-checking Engine of Rebeca Mohammad Mahdi Jaghoori," pp. 1810–1815.

[11] A. Gul, "A Foundation for Actor Computation," no. July 1993, 1993.

[12] A. Bertolino, G. D. Angelis, L. Frantzen, and A. Polini, "The PLASTIC Framework and Tools for Testing Service-Oriented Applications," pp. 106–139.

[13] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "WS-TAXI: A WSDL-based Testing Tool for Web Services," *International Conference on Software Testing Verification and Validation*, pp. 326–335, Apr. 2009.

[14] A. Bertolino, G. D. Angelis, L. Frantzen, and A. Polini, "Model-Based Generation of Testbeds for Web Services."

[15] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Whitening SOA Testing," pp. 161–170, 2009.

[16] L. Baresi, *Test and Analysis of Web Services*, 2007.

[17] G. Castagna, N. Gesbert, and L. Padovani, "A theory of contracts for Web services," *ACM Transactions on Programming Languages and Systems*, vol. 31, no. 5, pp. 1–61, Jun. 2009.

[18] A. Ferrara, L. Sapienza, and V. Salaria, "Web Services : a Process Algebra Approach," pp. 242–251.

[19] A. Textor, M. Schmid, J. Schaefer, and R. Kroeger, "SOA Monitoring Based on a Formal Workflow Model with Constraints," *Applied Sciences*, pp. 47–53, 2009.

Distributed Testing of Multicomponent Systems

Boris Tyutin, Igor Nikiforov, Vsevolod Kotlyarov
Saint Petersburg State Polytechnical University, Saint Petersburg, Russia
b.tyutin@ics2.ecd.spbstu.ru, i.nikiforov@ics2.ecd.spbstu.ru, vpk@ics2.ecd.spbstu.ru

***Abstract* — This paper features an approach that brings together testing of multicomponent systems, formal requirement specifications and automated test suit generation in a one technology.**

I. INTRODUCTION

In software development testing is usually performed at the end of the whole process. This traditional approach leads to the increase in the costs of correction of errors found during the testing or the program product piloting. This, together with the high resource-intensiveness of a testing process itself can lower the test coverage of software product and, therefore, its quality. This problem was partially solved by test-driven software development methods [1]. But the latter have strict limitations (related to the development process and coding politics), which make it unproductive to apply them for medium-size and large industrial projects. The main issue of these methods is the complexity of the adjustment and the scaling of software specifications. It is caused by the requirements modification and clarification during the lifecycle of the software product and it dramatically increases the time necessary for testing.

Thus, the possibility of using test suites related to different abstraction levels and their simultaneous execution gives the possibility to perform testing during all stages of the software product development – from the architectural design abstractions elaboration till product release. Testing automation in this case should significantly reduce the testing process costs.

In telecommunication software development it is a common practice to use a distributed architecture due to the peculiarities of the task domain. For such systems, test procedures have to imitate the whole environment or a part of it in order to be efficient. Also, the possibility of substituting some of the system components with test case entities makes it easier to locate an error or a "bottle neck" within the software. In this case, the testing laboratory contains the following parts:

- system under test (SUT);
- testing controlling system;
- monitoring and test data generation system;
- testing agents.

Distributed allocation and remote interconnection of different parts of the testing system add some important properties to the testing process:

- scalability;
- statistics collection and control of the testing process;
- test suite configuration and test result history storing.

This paper reviews the technology of automated software testing based on automated generation of the test suite from the formal specifications of the system in Use Case Maps notation. Also it briefly outlines the usage of symbolic verification technique (UCM Specification Translator or UST and VRS [6]). Finally, the architecture of distributed testing system (TestCommander) is described, which allows automatic test suite execution and adjustment according to the SUT architecture.

II. SYSTEM REQUIREMENTS FORMALIZATION

In order to be able to have an unambiguous interpretation of the requirements during the creation of a test suite, it is necessary to convert these requirements from an implicit or informal form to a notation, which provides semantic invariance when working with the specifications of the system. At the same time, if there are manual steps in the technology, the notation is required to be understandable by the user. The possibility of description of parallel interactions is also required in selected classes of testing tasks. The Use Case Maps (UCM) notation satisfies all these requirements. It was created by a group of scientists in the University of Ottawa [2] at the end of the twentieth century and is widely used for requirements engineering, system design and creation of test scenarios. UCM is used for behavior specification in telecommunication, distributed systems and other areas where it is necessary to specify an intercommunication of different entities. The notation has been standardized in 2003 by ITU-T [4]. The main elements of UCM notation are team (object or agent under observation), responsibility ( , action under observation), start point () and end point () of scenario, "and fork" ( ) and "and join" ( ) (parallel scenario description), failure point ( , exception under observation), timer ( , system timer object) and stub ( , is used for hierarchical diagram composition).

UCM project is a set of connected and structured maps containing a sequence of UCM elements. A set of maps describes the behavior of the system just as it is done in the requirements. The example of UCM map for a large telecommunication project is depicted in the Fig.1. There are three agents, one is for incoming request receiving (STMD), other is for request processing (RRt), and the third one performs exception handling (FRMP). In case of receiving of the incoming request (responsibility HandoffRecognized), the

timer (TStfHotDir) is set. If it expires, a corresponding timeout is fired (responsibility StmdHanUnsucc) and an exception occurs (failure point Failure). It is pretty easy to specify the behavior of a multicomponent system in terms of UCM agents and actions. But the way it is formalized depends on the experience and qualification of the engineer who creates the system specifications.

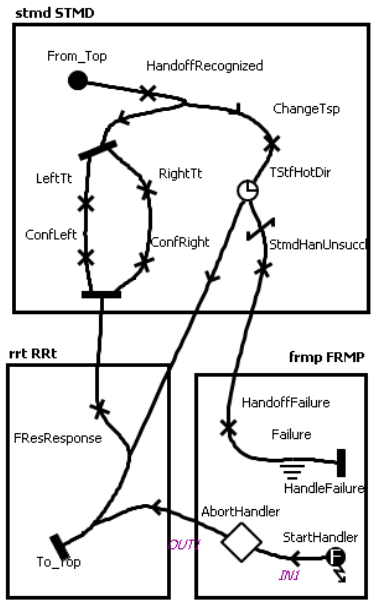


Figure 1. UCM map for a telecommunication project.

UCM notation allows defining system behavior in very convenient graphical way. In order to enable data flow, sending and receiving signal points definition (as well as usage of conditions for alternative behavior and exceptions), UST tool functionality was expanded by metadata grammar. The information from metadata is used during the process of formalization and is fully reflected in the model. Also, the ability to describe environment of the model, where the initial parameters are defined, was added.

For automatic transformation of high level design in UCM notation into formal description in basic protocol language UST [7] tool is used. Its main features are: the UCM analysis of errors, where the syntactic correctness of metadata and UCM structures are checked; the optimization of base protocol system model, which improves the efficiency of the instruments verification and testing; component-based approach of model structuring, which allows verification and test generation to be independent from individual parts of the model.

The tool helps to create the UCM model, add required information to metadata and translate it to base protocols (BP) [6]. One of the most important parts of the tool is UCM analysis module. Developing the UCM model, the user is guided by his understanding of the system requirements, his knowledge and experience in the behavior definition. But despite all the advantages of the developer, the model may have errors associated with user competence in formalization. The most common modeling errors are the ones of incorrect conditions of alternative branches and usage of shared resources by concurrent threads. The analyzer warns the user about potential dangerous places before building a formal model and helps him to make changes in order to remove the warnings.

Completed UCM project is checked for errors by the analyzer. After it is free of errors, it is possible to set up the formalization parameters. After the translation, the BP model is imported into the symbolic verification tool VRS. It performs the verification and the generation of test cases, which cover specified system requirements. Obtained symbolic test scenarios are filled with real data and can be used as a test suite specification for TestCommander tool or visual representation of system behavior.

III. DISTRIBUTED TESTING APPROACH

Full control over a test execution and a test result analysis are required to perform conformance testing of a system. Therefore, it is necessary to create a test suite, which is able to specify the interaction with the SUT in exactly the same way it is done in the requirements. Observing the behavior of the system during the execution of such test cases allows to establish whether the system implementation is correct or not[5].

For this purpose, Message Sequence Chart (MSC) language is used in TestCommander for test suite description [3]. It automatically generates the code of test suite modules on a target programming language (C++, Java, tcl) from MSC charts. These modules interact with SUT and each other using predefined interfaces, reproducing test scenarios. These interactions can be unambiguously and in a human-oriented manned defined in terms of MSC and, thus, represented in textual or graphical view.

The distributed testing method reviewed in this paper relies on automated approach for test suite generation. This approach is based on combined usage of UST and VRS tools and is demonstrated in Fig. 2.

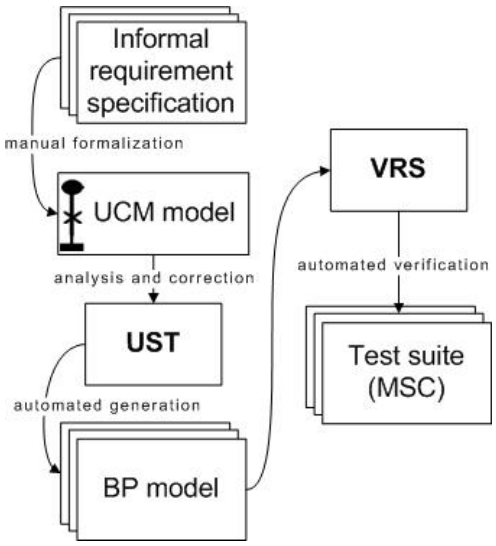


Figure 2. Automated test suite generation tool chain

As a combination of requirements formalization technique described earlier and the technology of test scenario generation based on the verification of a model in the BP notation [9], this approach brings together the requirements management, verification and testing in one technology. Automatically generated tests define the complete description of the interaction of all the components of SUT and its environment. All of the above allows these specifications to be tested and verified.

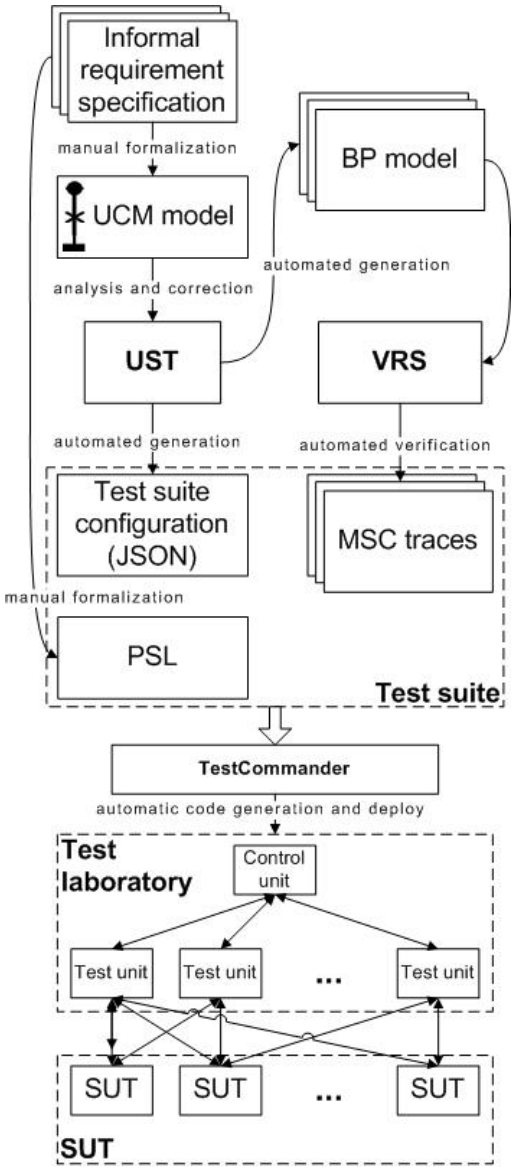


Figure 3. Distributed testing approach overview

TestCommander tool accepts MSC charts obtained with various methods (assuming that they correspond to MSC standard). However, the proposed approach has some significant advantages, such as:

- Automation of routine and time-demanding operations;
- Simple and human-oriented formal notations usage;
- Requirements checking with verification methods [6].

Generated test suite executable file set consists of one or more testing units and one control unit. Testing unit interacts with the SUT according to the test logic and exchanges the control signals with the control unit. The latter controls the test execution process and collects testing results. Protocols of interaction between testing system units and SUT are defined with Protocol Specification Language (PSL). This notation unambiguously specifies the format of the messages passed between the entities involved in the testing. PSL specification is created manually and is used for test suite code generation.

For test suite configuration, code generation and test suite deploy setup a configuration file is used. It is written in JSON [8] and specifies the location of testing units and SUT components. Nonetheless, the main feature of this configuration is to specify, which of the SUT components are substituted with the test units. It allows testing of the behavior only of a part of the system. The configuration file is generated automatically from UCM model, but it can be adjusted manually.

After the test suite is configured, its code in a target language is automatically generated and the test suite is deployed in the test laboratory. Testing starts with the execution of the control module of the test suite, which than controls the test unit threads and SUT. Test results are the MSC diagrams of test activities. The whole process is presented in Fig. 3.

IV. TECHNOLOGY APPLICATION LIMITATIONS

This approach is highly efficient due to the high level of the automation of routine stages of the test suite code creation and the adjustment of the whole test suite in case of any changes in the system specifications. However, as a result of some peculiarities of technologies included in the tool chain, there are limitations of its application in different tasks:

- Formal system specification can be created only in UCM;
- Generation of test cases is performed by VRS verification tool;
- Testing system has distributed architecture.

Pure UCM notation can be used only for functional requirements, which describe the control flow of the program. To alleviate this restriction, new semantic elements were introduced in UST tool – the metadata. It allows working with data flow of the system.

The verification tool VRS uses the state machine representation of the system in base protocol notation. The MSC diagrams are artifacts of the verification process and are used as test scenarios for test suite code generation. For some particular systems, it may take a significant amount of time to cover all the requirements with tests. As a solution for this problem, a new special feature was introduced in UST. By using the UCM model of the system, it can automatically generate special rules used in VRS during the analysis and trace generation. Also, a branch test criteria is used for the test suite optimization.

Another factor is the architecture of the testing system itself. It is based on the remote communication of the test units, the strict format of interconnection protocols and requires the pre-configured testing laboratory (linked workstations with some pre-installed software). Together with the state machine representation-based approach used in the test suite code generation, this allows the approach to be applied with the highest efficiency for the telecommunication system testing.

V. CONCLUSION

Multicomponent software testing problem is the complicated one. Computer-aided software engineering technologies can reduce efforts required for resolving this problem in industrial projects. Automation of different parts of software development process significantly increases the quality of the product. During our work a set of tools (UST, TestCommander) was developed. After integration with the verification system (VRS), it was used to form the semi-automated software testing method. The latter one was applied for various telecommunication projects (such as an implementation of a part of LTE standard).

The method, reviewed in this paper, is a combination of requirement management, verification and testing. It allows performing the checking of the correctness of the system implementation according to its specification within one technology. Testing approach is based on the automatically generated test suite, the correctness of which is proved during the system formal specification verification. It reduces the cost of regression testing needed in case of a changing or a refinement of specifications. All stages of this method are fully or partly automated. The developed software components used in these stages are independent; and all data formats are standardized. All of the above ensures that the whole method is scalable, highly flexible and adaptive and open for modernization. However, the technologies and implementation of the method limit the class of software testing problems, which can be successfully and efficiently solved with the approach reviewed in this paper.

REFERENCES

[1] K. Beck, Test-Driven Development by Example, Saint Petersburg: Piter, 2003.

[2] R.J.A. Buhr and R.S. Casselman, Use Case Maps for Object-Oriented Systems. London: Prentice Hall, 1996.

[3] ITU-T Recommendation Z.120: Message sequence chart (MSC). Geneva, Switzerland, October 1996, http://eu.sabotage.org/www/ITU/Z/Z0120e.pdf.

[4] ITU-T Recommendation Z.151 : User requirements notation (URN) - Language definition. Geneva, Switzerland, September 2003, http://www.itu.int/rec/T-REC-Z.151-200811-I/en.

[5] C. Kaner; J. Falk, H. Q. Nguyen. Testing Computer Software (2nd ed.). New York: Wiley, 1999.

[6] A. Letichevsky et al, “Basic protocols, message sequence charts, and the verification of requirements specifications”, Computer Networks: The International Journal of Computer and Telecommunications Networking, v. 49 n. 5, pp. 661-675, 5 December 2005.

[7] I. V. Nikiforov, A. V. Petrov, Y. V. Yusupov, “Generation of formal model of a system from requirements specified in USE CASE MAPS”, Scientific and technical statements STU. № 4 (103), pp. 191-195, St. Petersburg: St. Petersburg Polytechnic University, 2010.

[8] RFC 4627: The Application/JSON Media Type for JavaScript Object Notation (JSON), July 2006, http://www.ietf.org/rfc/rfc4627.txt

[9] A. O. Veselov, V. P. Kotlyarov, “Testing automation of projects in telecommunication domain”, Proceedings of the 4th Spring/Summer Young Researchers’ Colloquium on Software Engineering, Nizhny Novgorod, June 1-2, 2010, pp. 81-86.

An SDVRP Platform Verification Method for Microprocessor-Based Systems Software

Sergey Shershakov
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
Email: sashershakov@edu.hse.ru

Scientific Advisor: Prof. Irina Lomazova
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
Email: ilomazova@hse.ru

***Abstract*—The correctness of embedded systems software is of critical importance as invalid states can cause a physical damage to hardware. One of approaches to verification of such systems is using source code analyzers. The Static Driver Verifier Research Platform (SDVRP), which is based on Simultaneous Localization and Mapping (SLAM) and represents a tool that systematically analyzes source code and allows writing custom Specification Language for Interface Checking (SLIC) rules for various platforms, provided a potent verification mechanism for a thermal printer software system based on ARM Cortex-M0 microprocessor. An example of creating a custom platform plugin and rule verification is provided for the given embedded system.**

***Keywords:* Static Verification, SLAM, SDVRP, SLIC, Embedded Systems, ARM Microprocessor, Thermal Printer**

I. Introduction

In the early 2000s Microsoft seriously attended to issues of Windows drivers reliability. Thomas Ball and Sriram K. Rajamani, members of the Software Productivity Tools (SPT) research group of the Programmer Productivity Research Center (PPRC) at Microsoft Research, initiated a project called Simultaneous Localization and Mapping (SLAM) [1], the goal of which was to implement a rigorous performance test that some program is subject to the use rules of interface with which it interacts.

SLAM was based on the idea of using symbolic execution, model checking, and program analysis for proper interface use of programs in C language, which were represented by driver modules for Windows.

SLAM was the basis for the Static Driver Verifier (SDV), which is a compile-time static verification tool included in the Windows Driver Kit (WDK). The main purpose of the SDV is automated static testing of Windows drivers developed with the assistance of WDK tools. The SDV uses the "platform" ideology to determine how a driver must interact with a system (platform), which input-output interfaces to use and which methods of interaction with it to provide. To verify the interaction between a driver and a specific platform, a set of rules in the specialized SLIC language is developed.

The SDVRP (SDV Research Platform) is an extension of the SDV, the main difference with which is the ability to develop custom platforms and test drivers for them. The basis of the idea presented in this paper is speculation about the possibility of using SDVRP tools in an unusual way: for static verification of C programs for ARM microcontrollers (MCUs), wherein the MCU environment library is proposed to be used as a custom platform and a software module that implements a required functionality of MCU firmware is used as a test "driver".

The rest of this paper is organized as follows. Section II discusses the context in which the SLAM project took place. Section III discusses the Static Driver Verifier (SDV) tool and the SDV Research Platform (SDVRP) which is an extension to the SDV that allows adapting the SDV to support additional platforms for verification. Section IV is devoted to the research on application of the SDVRP for verifying MCU software (firmware) by the example of an embedded thermal printing system based on the ARM Cortex-M0 thermal microprocessor architecture. Practical application of the SDVRP for testing a thermal printing complex is discussed in Section V. Finally, Section VI concludes with an analysis of the work done and a look at the future.

II. SLAM

The main idea of SLAM, that is checking a simple rule for a complex program written in C (e.g., a driver), has to be realized by streamlining the program in order to make possible a comprehensive analysis. In other words, one needs a mechanism to obtain an *abstraction model* of the program which preserves the behavior of the original program in C.

The main question that arises at this point: „What form should take an abstraction model of a program in C?“ An approach based on *Boolean programs* was proposed.

A. Boolean Programs and "Bebop" checker

While shaping the idea of *Boolean programs* the developers were guided by the following characteristics, which a model checking tool should have [2]:

- for the modeled program there must be a representation of R analogous to a finite state machine (FSM)[1];
- a model checking algorithm in R should represent the shortest path to the model error, if any is found;
- there should be developed a mechanism of translating programs in high-level programming languages such as C, C++, Java to a model in R (abstracting);

[1]At that time the theory of finite state models verification was well-developed

- a specific model r in R can be refined to a model $r\prime$ in R and proved correct;
- the algorithms for model validation in R should support modularity and abstract constraints existing in the original program.

As a result, there has been developed an ideology of Boolean programs that have control structures common in imperative languages (such as C) and in which the only data type of all variables is Boolean. The Boolean programs contain procedures with parameters called by value, recursion, and control nondeterminism in a restricted form [3].

The Boolean programs are of interest for the following reasons.

1) The amount of memory (number of data cells), which the Boolean program operates, is finite (and even more so due to the limitation of the actual computing power), so the problem of accessibility and termination, that does not have a general solution, is soluble for Boolean programs (it is noted that the Boolean programs are equivalent in power to push-down memory automata generating context-free grammar).
2) Once the control structures of a Boolean program are similar to those of programs written in C, the Boolean programs are obvious candidates for testing programs with similar structures.
3) A Boolean program can be regarded as an abstract representation of a C program with available explicit correspondence between the data and control, and Boolean variables may represent some predicates on a generally unrestricted state of the C program.

To test Boolean program models a special model checker *Bebop* was developed. Formally, given a Boolean program B and an expression s in B, Bebop determines whether s is accessible in B. In other words, s is accessible in B if there exists an initial state of the program (as defined by its Boolean variables) such that starting execution of the program from this state, s will eventually be fulfilled. If the expression s is accessible, the shortest route to s is built.

B. Automatic Predicate Abstraction of C Programs

As noted earlier, the Boolean programs in their control structures are similar to programs in C. One of the main criteria considered in the development of Boolean program principles was the possibility of getting a model of a real C program in the form of its Boolean equivalent.

One of the most promising approaches which allows getting a Boolean model out of an original C program automatically was the principle of *predicate abstraction*, which consists in the following: the specific states of the system (source program) are mapped onto an abstract state in accordance with their assessment on a finite set of predicates.

In the continuation of the SLAM project there was developed a tool called *C2BP*, which performs such an operation [4]. Given a C program P, a set of predicates E, which are usual expressions in C without calling functions, *C2BP* automatically creates a Boolean program $BP(P, E)$, which is an abstract model of the program P. The Boolean program has the same control structure as the original program P but contains only $|E|$-Boolean variables, each of which has its own representation in the set of predicates E.

For example, suppose there is a predicate $(x < y)$ in a set E, where x and y are integer variables of a program P. Then there is a Boolean variable in a corresponding program $BP(P, E)$ such that if it is true at some point p of the Boolean program, it means that at this very point p of the original program P the predicate $(x < y)$ is estimated the true way.

For each expression s of the program P *C2BP* automatically creates a corresponding Boolean transfer function, which displays the effect of expression s on the predicates of E. The resulting Boolean program is to be evaluated by Bebop utility considered earlier.

III. Static Driver Verifier (SDV) and Static Driver Verifier Research Platform (SDVRP)

The SLAM analyzing mechanism became a core part of a new tool *Static Driver Verifier* (SDV), the main purpose of which is a systematic analysis of the source code for Windows drivers with the view of compliance with the set of rules defining how the drivers have to interact with the OS kernel [1].

A. Interface Temporal Safety Properties and Their Validation

An interface has a number of temporal safety properties which have to be respected by a program that uses the former; checking this is among the goals set by SDVRP developers. Safety means here that nothing abnormal occurs to the program. For instance, safety for a lock may signify that it has to be released only when it has been acquired [5]. Once a certain program has been attributed a safety property to comply with, it makes sense to verify the code does respect it, and if it does not, then to locate an execution path demonstrating how the program would break it.

The user does not have to supply abstractions or annotations (invariants) in order to validate or invalidate system software safety properties with the help of model checking. This implies that model checking is used for automatic computing of fixpoints over a C code abstraction expressed by a Boolean program. One constructs a proper abstraction in two steps: first, one gets an initial abstraction from the property that has to be validated, and then an automatic refinement algorithm is applied to refine the abstraction.

B. Specification Language for Interface Checking (SLIC)

There is a number of ways of how reliability of an APIs-based software system can be hindered: an API may not be correctly executed by an implementation L or the API may be treated wrongly by a client P [6]. Usually, only the API's documentation contains a set of requirements imposed on the client as well as on the implementer.

For specifying temporal safety properties of C language APIs a special low-level specification language *SLIC* was introduced to the SLAM project.

Suppose a state machine defined by a SLIC specification S monitors the behavior of a sequential composition $P||L$ of two programs P and L which happens at the API's procedural interface. This finite state automaton discards definite sequences of interface states of $P||L$ if the API is not properly implemented by L or P uses incorrectly the L-implemented API.

A SLIC *product construction* $Q\prime$ has an important property: it is such a product of a program $Q = P||L$ combined with the specification S that if and only if Q satisfies the specification S, a unique label (`SLIC_ERROR`) is not reachable in $Q\prime$.

C. Static Driver Verifier (SDV)

The SLAM tools enable verifying system software temporal safety properties completely automatically. Breakdowns are listed by the SLAM tools as paths over the program P. It automatically refines the abstraction by using the spurious error paths found.

The Static Driver Verifier (SDV), a tool that systematically analyzes the source code of Windows device drivers for compliance with the rules defining what it is to properly interact with the Windows operating system kernel, is essentially based on the SLAM analysis engine [1].

It contains, besides the SLAM engine, other constituents:

- a model of the Windows kernel and other drivers, called the *Environment Model* (Fig. 1);
- a large number of rules for the Windows Driver Model;
- scripts to build a driver and configure the SDV with driver specific information;
- a graphical user interface (GUI) to summarize the results of running the SDV and to show error traces in the source code of the driver.

The most important component of the SDV is the *Platform Model*, which represents an abstraction of the Platform.

The Platform Model includes three components (Fig. 2):

- a Platform Manager Model responsible for exercising the module by calling the module's entry points. It is in a way the entry point (main routine) of the system. For WDM drivers this is the Windows IO Manager;
- a set of Platform API Models that the module can use for completing requests from the Platform Manager. The APIs normally contain functionality to access the underlying hardware. In Windows this is the Windows Device Driver Interfaces (DDIs), consisting of hardware abstraction layer, APIs for controlling execution (locks, queues, etc.) and APIs for accessing resources such as memory;
- a Platform Model Infrastructure, which contains shared header files, common functions and states for the Platform Manager Model and the Platform API Models.

D. Static Driver Verifier Research Platform (SDVRP)

The SDV Research Platform (SDVRP) is an extension to the SDV that allows adapting the SDV to support additional platforms and writing custom SLIC rules for this platform [7]. Typically, driver platforms are the platforms one would adapt the SDV to verify, but it can also be any other module embedded to an OS or an application. If additional settings are made, one can use the SDV to check that API or protocol clients comply with the protocol/API specification.

IV. Background of using the SDVRP for verification of MCU software

A methodology for applying the SDVRP to verify embedded software systems based on an ARM Cortex-M0 microprocessor is hereinafter under consideration. This system is part of a hardware-software complex which performs the function of printing on heat-sensitive tape (thermal printing).

The correctness of this software is of particular importance in the sense that there are states of the program which can cause physical damage to hardware; exposing the program to such conditions is therefore unacceptable. For example, an incorrect sequence of pulses in the windings of the stepping motor can lead to mechanical jamming of the shaft, and an incorrect circuitry of the thermocouples can make them burn out and even make the print substrate go up in flames, which can lead to material damage and personal injury.

In light of these problems, the microprocessor firmware verification is important. This paper reflects the results of research on the use of SDVRP tools (positioned by the developers primarily as a tool for Windows drivers troubleshooting) for static verification of microprocessor firmware.

The key features allowing the use of the SDVRP for these purposes are considered as follows.

A. Source code language

The ARMs are RISC-microprocessors[2]. The MCU's instruction set is implemented in ARM assembler, which is the main programming language. At the same time, there are high-performance optimizing compilers for high-level languages C and C++, and the share of MCU programs developed in these languages increases every day, which is primarily due to an increase in MCU hardware resources.

The best is to use a C language compiler, which allows obtaining an equally efficient code, as if it were written in assembler. This allows, on the one hand, to use it in time-critical applications (which include the majority of embedded applications, including those for ARM MCUs), on the other hand, to use the entire power and convenience of a high-level programming language.

As it was shown in section II, SLAM, which underlies the SDVRP, allows automatically receiving predicate abstractions exactly for programs written in C.

It should be noted that the SDV imposes some restrictions on the code of the programs written in C which do not affect the expressive properties of the language but eliminate various tricky ambiguities. In C programs for microprocessor-based systems (in particular, those for C-to-ARM assembler compiler) an ANSI C superset with some additional constructs is used. Basically these are additional type identifiers and

[2]Restricted (Reduced) Instruction Set Computer

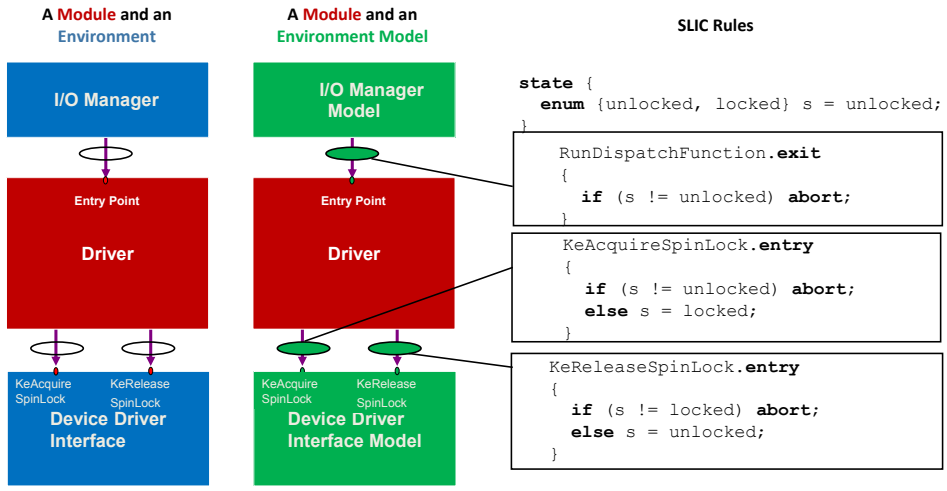


Figure 1: A Module and an Environment Which Are Substituted by an Environment Model and SLIC Rules

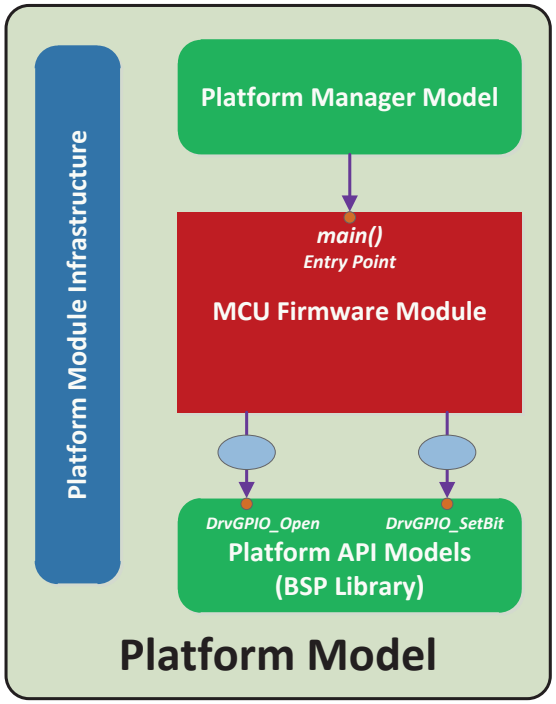


Figure 2: A Module and the Components of a Platform Model

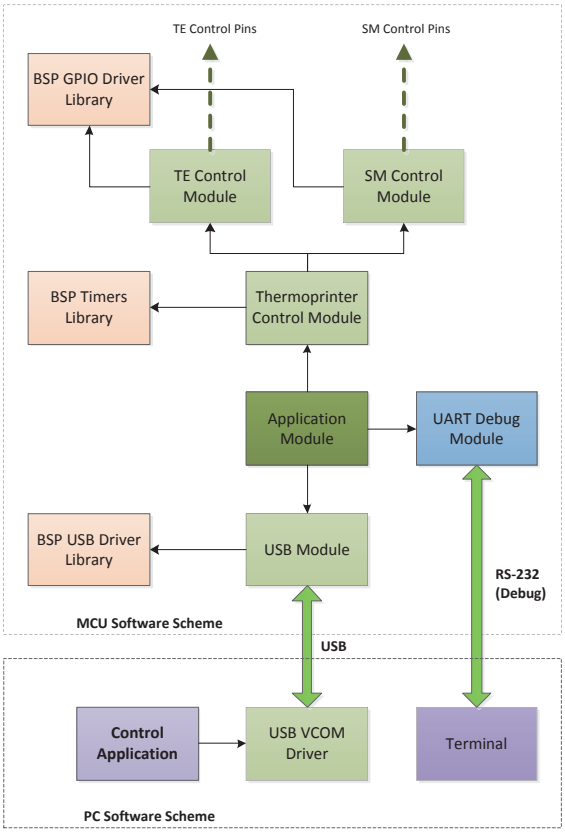


Figure 3: Thermal Printer Software Chart

keywords that govern the choice of memory to store the data. In order for the SDV to treat such input programs, the following workaround is proposed: to express such constructions in lexically similar directives of the C preprocessor, which are processed in due course while being compiled but ignored as irrelevant during verification.

The undoubted advantage is also the fact that the software library (the BSP library) for support of MP NUC140 (the ARM CMSIS for the kernel and Nuvoton Platform for the peripherals blocks) comes with the source code in two versions: in assembler and in C. The BSP library (refer to Fig. 2) corresponds to the Device Driver Interface Model in the diagram Fig. 1.

B. Modular structure of the program

Each of the units of the thermal head constituting the thermal printer has its own specific control which defines the rules for its programming: structures used, initialization

algorithms, etc (refer to SectionV). Each such unit has its own programming model expressed as a *function module*, which can be represented for clarity's sake as a separate translation unit — source file .c (and a corresponding header file .h).

The SDV verifies that a *module* interacts correctly with a *platform*. The platform is essentially a set of APIs, or a library. In this case the BSP library serves as a platform architecture and an ARM microprocessor.

The functional purpose and specificity of control of each unit is also a good source for developing SLIC rules which describe its specification. Formulation of SLIC rules that are specific for each module (and corresponding units) belongs is one of the main current tasks.

C. A look at the system as a whole

Another source for SLIC rules are BSP library program modules used jointly by several units/modules of programs to verify. An example of such a module is General Purpose Input-Output (GPIO) — routines to control the status of input-output pins for general purpose.

In general, access to shared resources is quite common while developing microprocessor-based software, which provides an opportunity for research on formulation of rules that control the correctness of such operations. The sequence of test calls is injected into the module under verification by an element called *harness*.

Positioned as an important property of the SDV is the fact that verification (while executing preliminary procedures such as writing SLIC rules, designing harness, etc.) is carried out automatically when building a driver project. Similarly, it is possible to use the SDVRP as an external batch process that is run by an IDE (e.g., Keil or IAR) while building the project. In addition, for translation of programs one can use compiler tools of these IDEs in conjunction with Eclipse, known for its scalable modular structure. This allows developing a special plug-in that performs verification on the basis of SDVRP modules controlled from Eclipse (this is a subject of future work).

Futher, a practical application of the SDVRP for testing a thermal complex is considered.

V. Device Software Verification: Study Case

The hardware-software complex considered previously (Fig. 3) includes:

- a Fujitsu-Siemens FTP-628MCL054 *thermal head*, which is a complete plug-in thermal printer module with physical layer interfaces to control of a stepping motor (SM), which feeds print substrate, and of low-inertia heating elements (TE) engaged in short-cycle heating-cooling in the printing zone (refer to the specification [8]);
- an *interface card* — a printed circuit board which was specially designed for this project and carries out conversion of control signal interfaces to physical interfaces of SM and TE control; the card also contains the necessary circuitry to support the functions of a comparator for paper detector and an ADC for the thermal sensor;
- a NU-LB_002 *debug board* — an evaluation board for developers of systems based on Nuvoton NUC140 microcontrollers, which provides physical access to the microcontroller's GPIO, CMP, ADC interfaces used to control the thermal head through the interface card;
- a *personal computer* (PC) which connects to the debug board via USB interface; another (special) interface is used for programming the microcontroller's USB ICE; debugging information from the MCU connected to the debug board is obtained via RS-232 interface.

The thermal head consists of the following functional units (refer to [8]):

- stepping motor;
- thermal head;
- thermal head temperature detector (thermistor);
- paper detector – mark detector (photo interrupter);
- platen release (platen open switch).

These units are controlled independently from each other through the formation of a sequence of analog-digital signals and analysis of responses received. In most cases, to perform functionally complete operations of the head (such as printing, feeding tape, etc.) coordinated participation of several units (e.g., the thermocouples and the stepping motor) is required. Control, coordination and analysis of the current state of the thermal head and the supporting elements (parts of the interface card) is carried out using the software (firmware) recorded in volatile memory of the MCU (flash memory).

A following problem of MC software correctness is considered below. Nu140-family MCU has a large number of pins, which can be used by the developer to control a device based on this MCU. These include the so-called GPIO pins (grouped in a few ports), which have various types of circuit implementation, depending on the task. Modern circuitry allows selecting an operation mode of each pin at the software level, by setting a value of the corresponding configuration register. In some tasks it is critical to correctly set an operation mode of a pin *before* it is first used, as otherwise the pin port and the net attached to it may be damaged. This way the *correctness of MCU software* implies a code that is guaranteed to correctly initialize a pin before its first use.

Below is shown how one can validate MCU software with the help of the SDVRP.

A. Custom Platform Plugin

Files that belong to a certain platform can be organized into a "plugin" if the SDV and SDVRP are implemented. A plugin for a custom program should be created in order to apply the SDVRP to that platform.

Let the test plugin be called *XiPlatform1*. Physically, three subdirectories in the SDVRP directory correspond to it; the path to the SDVRP directory is given by the environment variable `%SDV%`. The directory `%SDV%\data\XiPlatform1` is for configuration files, `%SDV%\rules\XiPlatform1` — for rules files and `%SDV%\osmodel\XiPlatform1` — for the harness and other auxilary files.

Platform API Models on the diagram in Fig. 2 is a software environment which is called by the verified module. With respect to the system under consideration such a program environment is the BSP Library supplied by the MCU manufacturer and ARM core licensee. Finally, the verified MCU software itself corresponds to the MCU Firmware Module element.

B. Rule verification

Pin initialization is done by calling a BSP Library function `DrvGPIO_Open` with corresponding parameters. In this example, `ENA` pin enabling the SM driver microchip, the state of which is subsequently changed in a certain sequence by calling the functions `DrvGPIO_SetBit` and `DrvGPIO_ClrBit`, is initialized.

A rule that checks that `ENA` pin is initialized before its first use looks as follows:

```
#include <XiGPIOUse_slic.h>
state{
    enum {closed, opened} enpinout = closed;
}
DrvGPIO_Open.entry
{  if($1 == TSM_PORT_EN && $2 == TSM_PIN_EN)
      enpinout = opened;
}
DrvGPIO_ClrBit.entry
{  if(enpinout == closed)
      abort "The driver is calling $fname
               before the pin is opened.";
}
DrvGPIO_SetBit.entry
{  if(enpinout == closed)
      abort "The driver is calling $fname
               before the pin is opened.";
}
```

The algorithm that calls pin initialization and control methods is either a separate algorithm or a part of another algorithm. The call of this algorithm is carried out in a routine which is the entry point of the module and corresponds to the function `main` of the C program. This routine is part of the Platform Manager Model harness and described in a special way.

VI. Conclusion

In this paper there was given a review of SLAM tools and the potential of SDV tools based on the former, as well as that of SDVRP version ones. There was revealed a possibility of using SDVRP tools for static verification of embedded microcontroller software system that use source codes in C language by the example of a control system of thermal printing based on a NU140 microprocessor with the ARM Cortex-M0 core. Among the tasks of current importance there are the completion of a SDVRP plug-in which implements the Environment Model for NU140 and the BSP library, as well as the development of the necessary SLIC rules for modules of the embedded system and their verification on the SDVRP platform.

The considered study case (creating a custom platform plugin and rule verification) is part of a larger verification system and presented in a nutshell, due to limitations on paper length which do not allow more scrutiny.

Acknowledgment

The author would like to thank Prof. Irina A. Lomazova for her vital encouragement and support.

References

[1] T. Ball, B. Cook, V. Levin, and S. K. Rajamani, “SLAM and static driver verifier: Technology transfer of formal methods inside microsoft,” *IFM*, pp. 1–20, 2004.

[2] T. Ball and S. K. Rajamani, “Boolean programs: A model and process for software analysis,” Microsoft Research, Tech. Rep., 2000.

[3] ——, “Bebop: A symbolic model checker for boolean programs,” *SPIN 00: SPIN Workshop*, pp. 113–130, 2000.

[4] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani, “Automatic predicate abstraction of C programs,” *SIGPLAN Not.*, vol. 36, no. 5, pp. 203–213, may 2001.

[5] T. Ball and S. K. Rajamani, “Automatically validating temporal safety properties of interfaces,” pp. 103–122, 2001.

[6] ——, “SLIC: a specification language for interface checking (of C),” Software Productivity Tools, Microsoft Research, Tech. Rep., 2002.

[7] *Static Driver Verifier Research Platform. Introduction (sdvrp.docx)*.

[8] *Fujitsu Takamisawa Component Ltd. Thermal Printer FTP-628MCL054. Product Specification*, 2000.

Instantiation-Based Interpolation for Quantified Formulae in CSIsat

Vadim Mutilin
Institute for System Programming, RAS
mutilin@ispras.ru

Mikhail Mandrykin
Institute for System Programming, RAS
mandrykin@ispras.ru

***Abstract*—The paper describes an implementation of instantiation-based interpolation for quantified formulae in modified CSIsat tool. The tool supports interpolation for formulae with linear real arithmetic, uninterpreted functions and quantifiers. We propose in this paper using external SMT-solver CVC3 for quantified expressions instantiation, then we describe how we modified CSIsat and CVC3 tools in order to support quantified formulae interpolation. We also present results of benchmarking the modified CSIsat tool on SMTLIB test set as well as on our specially generated interpolation tasks.**

***Index Terms*—interpolation, Craig interpolant, quantifiers, instantiation, solver, axioms.**

I. Introduction

Among several currently predominant model checking techniques, predicate abstraction is one of the most widespread and successful approaches. This success is significantly promoted by the advance of SLAM2 [1] static verification tool, which uses this approach and which is extensively used in WDDK(Windows Driver Developer's Kit) tool set for static verification of windows device drivers. Program predicate abstraction is built with logical predicates. The set of all possible valuations of the predicates from the abstraction forms an abstract domain partitioning the total program state space into subsets with same predicate valuations[14]. The particular challenge here in predicate abstraction is identifying necessary predicates, since they determine the accuracy of the abstraction. In most state-of-the-art predicate abstraction tools predicate choice is fully or mostly determined by the program considered. So one of the most important problems arising with the use of predicate abstraction is the derivation of the predicate set suitable for the verification of each particular program. Most frequently used decision for that problem is using the CEGAR approach [12] — Counter-Example Guided Abstraction Refinement. The approach is based on the iterative construction of the program abstraction starting with the most coarse one and continuing consistently with one or several successive abstraction refinements based on corresponding infeasible counterexamples that arise from the verification of the coarser abstraction. So when the coarser abstraction gives a spurious (infeasible) counterexample, the fact of its infeasibility is somehow used for refinement of that inaccurate abstraction.

This work was partially supported by FTP "Research and development in priority areas of scientific and technological complex of Russia in 2007-2013" (contract number 11.519.11.4006)

Among the modern static program verification tools which use predicate abstraction and implement the CEGAR approach, SLAM2[1], BLAST[7], [23] and CPAchecker[8] are the most extensively used in practice. These tools implement abstraction refinement in two rather similar and still a little different ways. All the tools somehow build a logical formula corresponding to the counterexample considered. SLAM2 uses weakest precondition for the statement sequence of the counterexample. BLAST and CPAchecker both build a path formula based on the SSA (Static Single Assignment) form. As soon as the counterexample is spurious, its weakest precondition and a path formula are unsatisfiable, if build precisely enough. This fact is used by the tools to derive new predicates and refine current abstraction. Here software model checkers bring some special tools into play. SLAM2 uses heuristic predicate derivation from the *unsatisfiable core* (i.e. a small unsatisfiable subset of clauses) of the counterexample's weakest precondition. The satisfiability check and unsatisfiable core extraction are performed by special tool called *SMT-solver* (SMT stands for Satisfiability Modulo Theories) intended to decide logical formulae with respect to combinations of background theories expressed in classical first-order logic with equality. SLAM2 uses Z3[13] SMT solver. Other syntax-based predicate derivation techniques[16] are used in SLAM2 as well. BLAST and CPAchecker derive new predicates locally for selected program locations (such as loop heads, functions calls or just any program statement) from *Craig interpolants* of the two path formula parts before and after each of such a location point. A Craig interpolant for a mutually inconsistent pair of formulae (A, B) is a formula that is implied by A, inconsistent with B, and expressed over their common uninterpreted symbols (variables and uninterpreted functions occurring both in A and in B). To find Craig interpolants both BLAST and CPAchecker use special *interpolating SMT-solvers*. An interpolating SMT-solver extends a decision procedure by taking a conjunction of a pair of logical formulae as an input and by producing one of the two possible results: either answer `SAT` — if the input conjunction turns out to be satisfiable — or a Craig interpolant for that conjunction (since in this case the conjunction is inconsistent and the corresponding Craig interpolant always exists by the Craig interpolation theorem). BLAST can use either FOCI[18] or

CSIsat[9] interpolating solvers and for CPAchecker the same CSIsat and MathSAT[10] are suitable.

The predicates derived form the counterexample analysis are intended to eliminate the spurious execution path form the abstraction and thus prove the absence of corresponding particular error. And when the predicates obtained from Craig interpolation of the path formula are guaranteed to rule out the infeasible counterexample by definition (provided that the obtained interpolants are inductive, i.e. each subsequent interpolant is implied by the conjunction of the previous one with the corresponding path formula part), the predicates produced from the heuristic approaches not necessarily succeed. Another advantage of interpolation as a technique for refinement is that it not only discovers new predicates, but also determines the control locations at which these predicates are useful. At the same time syntax-based predicate derivation approaches are also complete for certain classes of programs and their actual implementations quite rarely fail to discover necessary predicates.

One of the most significant SLAM2 advantages over current versions of BLAST and CPAchecker tools is its good precision (which is judged by the number of given false alarms, i.e. spurious counterexamples the tool considers feasible) in analysis of programs with significant use of pointers, including the ones to dynamically allocated memory regions. Meanwhile the tool doesn't use any special analysis for dynamically allocated data structures (like Shape-analysis[6]) as well as any special background logical theory for heap objects representation (e.g. separation logic [22]). A simplified location-based logical memory model is used instead. This model is proposed in paper [2] to be used for efficient evaluation of pointer predicates with a modern SMT solver (such as Z3). The paper authors suggest using the special axiom set and representing pointer predicates with uninterpreted functions of integer values.

We decided to investigate the possibility of using some similar logical memory model in BLAST or CPAchecker. As we have already stated above, the new predicates derivation in these tools is based on Craig interpolation of several parts of the unsatisfiable path formula [15].

The conjunction of the ordered pair of formulae corresponding to the two parts of the infeasible error path is unsatisfiable. Hence there exists a Craig interpolant for the pair. The predicates occurring in the interpolant are used by BLAST and CPAchecker tools in refining current program abstraction in order to eliminate the infeasible counterexample. An interpolating decision procedure (also called "interpolating prover", "interpolating solver" or just "an interpolator") is used in obtaining the interpolant. Both BLAST and CPAchecker use interpolating solvers (CSIsat and MathSAT) for quantifier free fragments of the logical theories of linear real arithmetic (LA) and uninterpreted functions with equality (EUF). The solvers are able to find quantifier free interpolants for such formulae. When using a logical memory model with an axiom set we need also to find interpolants for quantified formulae. At the same time, the getting of quantified interpolants for such formulae is also acceptable.

This way we see that SLAM2 with logical memory model for pointers needs only a good SMT solver with the support of quantifiers and unsatisfiable core extraction. But an implementation of a similar model in either BLAST or CPAchecker tool would essentially need an interpolating decision procedure for quantified formulae. But even after a thorough search by the time of our investigations we hadn't found any suitable tool for that purpose.

At the same time a paper [11] presents an extension of McMillan's algorithm[18] for instantiation-based quantified formulae interpolation. The formulae may be given with respect to an arbitrary combination of background theories for which the original McMillan's algorithm is applicable. The extended McMillan's algorithm gives us a rather simple way to find possibly quantified interpolants for such formulae in case the set of required quantified expression instantiations is known in advance. This instantiations must be sufficient for proving the given input conjunction unsatisfiable.

The idea of implementing the extended McMillan's algorithm also appeared to be interesting in account of the fact the verifiers usually perform several corresponding SMT solver queries just before the interpolation. And the majority of modern SMT solvers implement quantified formulae interpolation support through the instantiation of quantified subexpressions. This means that if the solver finishes with an `UNSAT` result providing a proof of the input formula unsatisfiability, the interpolating solver will that way have the necessary instantiations in advance. So long as the extended McMillan algorithm's implementation in case of a priori given necessary quantifier instantiations is reasonably easy, this implementation might be as well used for preliminary benchmarking the logical memory model efficiency in static software verifiers using interpolation for abstraction refinement. This way we decided to implement the extended McMillan's algorithm based on some existing interpolating prover and an SMT solver with quantifier support. We also decided to estimate the efficiency of the new tool on specially generated benchmarks simulating the interpolation tasks a real model checker could give to our tool.

II. Our approach

To apply an extended McMillan's algorithm proposed in [11] we need a resolution proof of input conjunction unsatisfiability with necessary quantified expression instantiations used and so-called partial interpolants in the leafs of the tree evaluated. The tree can be obtained in several ways. One such way is to implement some quantifier instantiation heuristic (e.g. e-matching[20]) in a tool currently implementing original McMillan's interpolation algorithm. The other way is to use an external tool successfully implementing such heuristics, say SMT solver. In this case the necessary instantiations can be extracted from the unsatisfiability proof given by the solver.

If we only extract necessary instantiations from the proof, then the formula unsatisfiability will be discovered twice: once by the SMT solver supporting quantifiers (to obtain

instantiations) and then again by the interpolator, here with necessary instantiations and without quantifiers, — to extract the desired interpolant from the proof. Meanwhile we can't use the unsatisfiability proof from the SMT solver for the interpolation directly, as according to McMillan's algorithm we need a specific unsatisfiability proof using inference rules significantly different from the ones used in modern SMT-solvers. Yet the state-of-the-art SMT solvers are significantly more efficient than any of the interpolating decision procedures we knew to implement McMillan's algorithm (they were FOCI, CSIsat and several experimental implementations). So when using an external SMT solver the overhead of proving the formula unsatisfiable once again is more on the interpolator's side.

Despite this significant overhead arising from using an efficient SMT-solver together with considerably less efficient interpolating decision procedure we eventually decided to implement the approach due to its relative simplicity. For that we had to choose a suitable existing interpolating solver meeting the following requirements:

- The solver should implement the McMillan's interpolating algorithm form the paper [18]. This was required as the algorithm presented in [11] is an extension of this McMillan's algorithm.
- The solver was required to support interpolation for logical formulae with respect to the combination of theories used by verification tools BLAST and CPAchecker in their interpolation queries. These are the theories of linear integer (LIA) and real (LRA) arithmetics and the theory of uninterpreted functions with equality (EUF). Their combination is often referred as LA+EUF.
- The source code of the solver should be freely available for modification and the solver should be distributed under an appropriate license.

Among existing interpolating decision procedures we knew and considered the only one to meet all the requirements was CSIsat[9]. It's implemented in OCaml and its components responsible for reading an input formula, preprocessing it, deciding its satisfiability, generating partial interpolants and combining them are placed in several fairly independent modules. So it turned out we only needed to implement modified versions of some of the modules and then use them whenever an input formula included some quantifiers.

Our relatively simple approach chosen used only a set of necessary quantifier instantiations and had no need in thorough analysis of unsatisfiability proof produced by the SMT solver. So the primary criteria for the choice of an SMT solver were its support of quantifiers, high performance and also, preferably, the availability of its source code for easy integration. Based on the criteria we choose CVC3[5] SMT solver. There the most relevant issue of the solver was the use of quite many complicated and coarse-grained inference rules in its proofs. The issue seemed to be minor at the moment as we only needed to extract relevant quantifier instantiations and not to process the entire proof. But later it turned out that sometimes necessary instantiations are not included in the final proof by the solver, which causes the modified interpolator to terminate abruptly.

III. Related work

The significant overhead of proving the input conjunction unsatisfiable the second time in the interpolating solver suggests the idea of another interpolation technique. The transformation of the SMT solver proof tree into the inference system appropriate for inductive interpolant derivation is the another approach essentially different from the one proposed in this paper. This approach is also greatly differs from both the one used in CSIsat and the one described in the paper [11]. The approach is reported in paper [19] and quite possibly performs much better than the one we consider here. The paper [19] proposes this approach for the Z3 SMT solver. Its application for another common SMT solver for quantified formulae — CVC3 — is complicated with great number of inference rules used by the tool. We ought to mention also that the paper mentioned was published a couple of months after we had finished implementing the tool presented in this paper. We only state the implementation details and efficiency benchmarking results relevant to our modified version of CSIsat tool onwards.

IV. Implementation details

The modified CSIsat tool supports interpolation of quantified formulae with respect to the combination of the two background theories: the theory of linear real arithmetics and the theory of uninterpreted functions with equality (LA+EUF). Our implementation is based on the latest version of the tool from its original developers, CSIsat 1.2 dated back to July, 2008. The tool uses the patched version of CVC3 SMT solver permitting easy extraction of necessary quantifier instantiations. Let us itemize the modifications we made upon the CSIsat and CVC3 tools in order to implement our approach.

A. CSIsat tool modifications

- The modified version of CSIsat tool has the extended input formula format compatible with the one used in the FOCI interpolating solver (implementing the approach presented in [18]). The input formula syntax has been extended with the designations for existential and universal quantifiers.
- The modified tool supports interpolation for pairs of formulae only. With the use of an extra option one can specify three input formulae: A, B and C. The formula C must be a conjunction of universally quantified expressions. In this case the interpolant is produced for the pair of formulae $(A, B \wedge C)$, but the free symbols from the formula C are considered to be common for the formulae A and $B \wedge C$. Here universally quantified expression from C are in this way considered somewhat like theory axioms.
- We implemented some preliminary transformations of the input formula before reducing it into the conjunctive

normal form. Each formula of the input problem is subject for the following transformations:

  - selection of the topmost quantified subexpressions,
  - reduction of the selected subexpressions into the prenex normal form,
  - *skolemization* of the subexpressions.

  Skolemization is a way of removing existential quantifiers from a formula. Variables bound by existential quantifiers which are not inside the scope of universal quantifiers are simply replaced by constants. And when the existential quantifiers are inside the universal quantifiers, the bound variables are replaced by Skolem uninterpreted functions of the variables bound by the universal quantifiers.
  If after the transformation the formula still includes at least one universal quantifier the extended McMillan's interpolation algorithm is applied. Otherwise the original algorithm is used.
- The support for SMTLIB v.2 [4] as output format was added. In case of quantified formula interpolation the transformed conjunction $A \wedge B$ is passed to the modified CVC3 SMT solver. If the formula is proven unsatisfiable, the modified tool also produces the set of essentially used quantifier instantiations. If the conjunction turned out to be satisfiable, both CVC3 and CSIsat terminate with `Satisfiable` verdict. As the SMT solver is incomplete in presence of quantifiers the `Unknown` verdict is also possible.
- The algorithm of *purification* of the essential quantifier instantiations from the *mixed terms* accordingly to the algorithm presented in [11] was implemented. If the conjunction is proven unsatisfiable, CSIsat purifies the obtained instantiations building auxiliary hash tables that contain the information about common symbols and newly introduced variables. It also computes the reflexive transitive closure of the inverse of the *support* relation over the newly introduced variables using the Warshall-Floyd algorithm.
- The extended McMillan's interpolation algorithm for quantified formulae was implemented. The algorithm works upon the proof tree obtained from CSIsat internal SMT solver together with the purified instantiations and the auxiliary hash tables generated on the previous steps. The algorithm may in general produce a quantified interpolant.

B. CVC3 tool modifications

The interpolation implementation requires the solver to produce necessary quantifier instantiations whenever it ends up with an `Unsatisfiable` verdict. To obtain the instantiations the generated unsatisfiability proof with explicit quantifier instantiation inference rules may be used. The CVC3 inference system has four such instantiation rules:

$$\frac{T \vdash \forall \bar{x}.e(\bar{x})}{T \vdash e(\bar{t})}\ universal\_elimination1$$

$$\frac{T \vdash \forall \bar{x}.e(\bar{x})}{T \vdash \psi \implies e(\bar{t})}\ universal\_elimination2,3$$

$$\frac{T \vdash \forall \bar{x_1}\bar{x_2}.e(\bar{x_1},\bar{x_2})}{T \vdash \psi \implies \forall \bar{x_2}.e(\bar{t},\bar{x_2})}\ partial\_universal\_instantiation$$

where $\bar{x} = (x_1, ..., x_n)$ is a bound variables vector, e is an expression in which the variables $x_1, ..., x_n$ occur free, $\bar{t} = (t_1, ..., t_n)$ is a vector of substitution terms, whose variables must occur free in $\forall \bar{x}.e$. For the $universal\_elimination1$ rule each x_i and t_i for every $i = \overline{1,n}$ must be of the same type. For other rules they should have the same basic types and in this case ψ is a predicate restricting the possible valuations of the substituted terms (to the subtype domain).

But it appeared in practice that CVC3 doesn't always include these inference rules into the final unsatisfiability proof, but sometimes replaces the branches emerging from quantifier instantiations with a much simplified inference rule of the form:

$$\frac{}{T \vdash e(\bar{t})}\ assump$$

Therefore we implemented a heuristic considering some unfinished proof tree fragments in search for the occurrences of quantifier instantiations. We implemented an instantiation simplification heuristic using the CVC3 internal expression simplification capabilities as well. The heuristics are switched with corresponding options.

V. Results

A. SMT-LIB benchmark set results

The modified tool has been tested on two distinct benchmark sets. The first one was obtained from the SMT-LIB[3] benchmark set by dividing the unsatisfiable formulae from the AUFLIA (AUFLIA stands for Arrays, Uninterpreted Functions and Linear Integer Arithmetic) and AUFLIRA (AUFLIRA stands for Arrays, Uninterpreted Functions, and Linear Integer and Real Arithmetic) logics randomly into two sub-formulae at the top-level conjunctions. The interpolation problems were translated into the modified CSIsat input format. Here all integer variables and functions were replaced with real ones and array operations (i.e. `select` and `update`) were represented using uninterpreted functions with the appropriate axiom set. Some of the problems became ill-posed (as the conjunction turned satisfiable) after the transformation and some other yielded degenerate interpolants e.g. `true` and `false`. The benchmarking was performed on the binary optimized version of the tool with the time limit of 5 s and the memory limit of 1 GiB. The results of the benchmarking in both the categories (AUFLIA and AUFLIRA together) are presented in table I.

TABLE I
SMT-LIB BENCHMARK SET RESULTS.

	Results	Number of tests	%
65.8% OK	Quantified interpolant	167	0.63%
	Ground interpolant	551	2.09%
	Degenerate interpolant `true`	8259	31.38%
	Degenerate interpolant `false`	8342	31.70%
	Satisfiable input conjunction	1	0.00%
34.2%	CVC3 gave `Unknown` verdict	536	2.04%
	Time limit exceeded (5 s)	7846	29.81%
	Memory limit exceeded (1 GiB)	71	0.27%
	Insufficient instantiation set	515	1.96%
	Miscellaneous errors	31	0.12%
	Total	26319	100.00%

B. Performance on specially generated benchmarks

The second benchmark set was generated as a collection of specially made-up simulated interpolation tasks a real model checker could give to our modified CSIsat tool in case it implemented a logical memory model similar to that of SLAM2 tool. The location-based logical memory model for pointer predicate derivation uses the following five uninterpreted functions:

- $A(l)$ — returns the address of the location l,
- $L(a)$ — returns the location corresponding the address a,
- $S(x, f)$ — returns a location for a composite type field or an array element, here x is the location of the composite type (or array) as a whole and f is the desired field number (or array index), counting from 0,
- $B(l)$ — returns the location of the a composite type (or an array) by the location of its element (l),
- $O(l)$ — returns the composite type field number by its location (l).

Here is the corresponding axiom set:

$$\forall x.(x > 0 \implies A(x) > 0) \tag{1}$$
$$\forall l.L(A(l)) = l \tag{2}$$
$$\forall a.A(L(a)) = a \tag{3}$$
$$\forall x.\forall f.S(x, f) > N \tag{4}$$
$$\forall x.\forall f.B(S(x, f)) = x \tag{5}$$
$$\forall x.\forall f.O(S(x, f)) = f \tag{6}$$

Locations here are denoted with strictly positive integer numbers. The first axiom states that the address of every normal basic location is strictly positive. The second and the third axioms together specify the functions $A(l)$ and $L(a)$ as mutually inverse for all the locations and their corresponding addresses. The fourth axiom states that the location of the composite type field (or an array element) do not coincide with any basic location. Basic locations correspond to explicitly allocated memory objects such as variables, arrays and structures as a whole. They all have location numbers in the range from 1 to a certain constant N. The composite field locations on the other side must have location numbers strictly greater than N, which is stated in the axiom (4). The axioms (5) and (6) specify the functions $B(l)$ and $O(l)$ and state that the elements of distinct composite values correspond to the distinct locations. Otherwise we'd get:

$$\begin{gathered} S(x_1, f_1) = S(x_2, f_2) \implies \\ B(S(x_1, f_1)) = B(S(x_2, f_2)), \\ O(S(x_1, f_1)) = O(S(x_2, f_2)) \implies (5, 6) \implies \\ x_1 = x_2, f_1 = f_2 \end{gathered}$$

To denote the location values we used a set of uninterpreted functions (V_i). Each such function corresponded to a state of the whole program memory between its two sequential updates. Here we didn't anyhow optimize the memory updates, so each of them gave an expression of the following form:

$$V_{i+1}(l_{upd}) = v \wedge \forall l.(l \neq l_{upd} \implies V_{i+1}(l) = V_i(l))$$

where l_{upd} is the number of the location whose value is updated to v.

The benchmarks making significant use of structures and arrays were performed with a slightly modified axiom set to take the structure and array first element address property into account. The address of a structure (or an array) is equal to the address of its first field (or element). So for such benchmarks we changed the axioms (4), (5) and (6) with the following ones correspondingly:

$$\forall l.A(l) = A(S(l, 0))$$
$$\forall x.\forall f.(f \neq 0 \implies S(x, f) > N) \tag{7}$$
$$\forall x.\forall f.(f \neq 0 \implies B(S(x, f)) = x) \tag{8}$$
$$\forall x.\forall f.(f \neq 0 \implies O(S(x, f)) = f) \tag{9}$$

Here the location of a structure or an array as a whole is merged with the address of its first field. The axioms (4), (5) and (6) are restricted to all the fields of an aggregate except the first one.

Let's illustrate the process of converting the pointer predicate derivation problem into the one of interpolating the quantified formula for one particular cut-point of an infeasible program error path. Here we use the following example from our benchmark set:

```
s_1->f_1 = 1;
s_1 = s_1->next;
s_1->f_1 = 2;
s_1 = s_1->next;
...
s_1->f_1 = n - 1;
s_1 = s_1->next;
s_1->f_1 = n;
s_2 = s_1;
s_1 = s_1->next;
-----------------------
check (s_2->f_1 != n);
ERROR:
```

Here n is a static parameter assigning the size of the test generated. When $n = 2$ the test will look like this:

```
s_1->f_1 = 1;
```

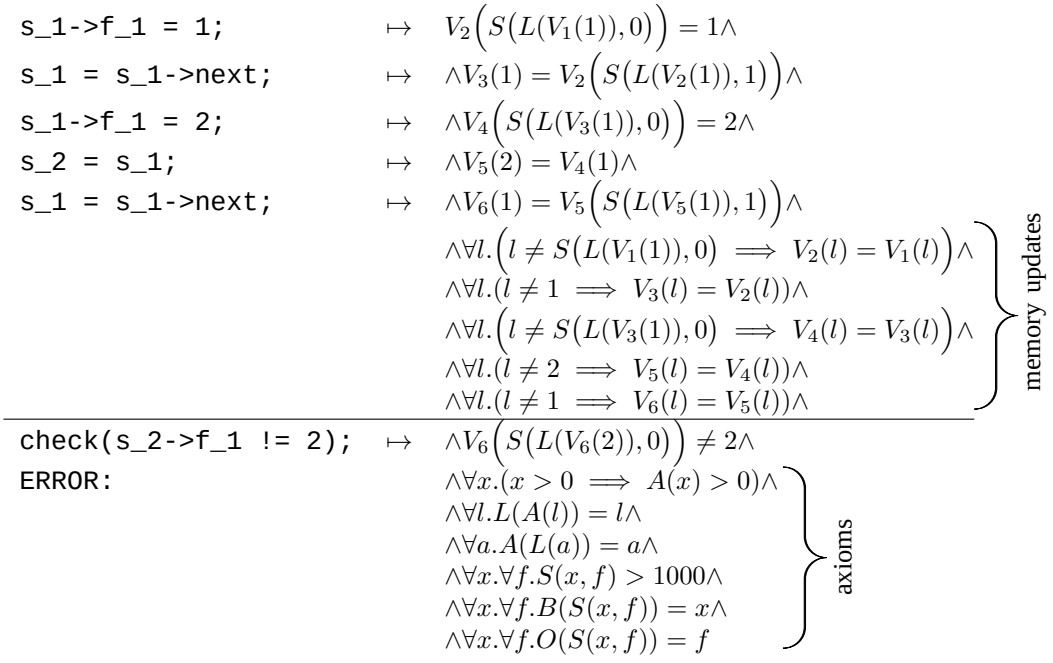


Fig. 1. Path formula example.

```
s_1 = s_1->next;
s_1->f_1 = 2;
s_2 = s_1;
s_1 = s_1->next;
----------------------
check(s_2->f_1 != 2);
ERROR:
```

Here we traverse a linked list of at least two elements and sequentially assign the values 1 and 2 to its elements with the help of the pointer `s_1`. The pointer `s_2` is assigned the address of the second element of the list. The `check` operator is used to designate the chosen branching condition. This means that the condition in the operator must be met on the considered error path. This way the necessary condition of getting onto the error label `ERROR` is the inequality of the second list element value to 2. It can't be fulfilled in case the program behaves correctly in terms of memory operations. The new pointer predicate proving the error path infeasibility should be derived for the cut-point marked with the dashed line separator.

Both the upper and the lower parts of the counterexample path correspond to certain logical formulae. For the newly derived predicates to make the abstraction eliminate the infeasible counterexample, the predicates must be implied by the logical formula for the upper part of the path and be unsatisfiable in conjunction with the formula for its lower part. Also to be possibly used again to eliminate other similar error paths the predicates should include only the values the program variables possess in the cut-point (not before and not after it). Otherwise it won't be possible to make the new abstraction independent from the currently considered counterexample. These requirements for the derived predicates exactly match the definition of the Craig interpolant for a pair of logical formulae, i.e. a third formula that is implied by the first formula, inconsistent with the second one, and expressed over their common uninterpreted symbols (variables and uninterpreted functions occurring in both formulae). Let's consider the formulae and the interpolant produced by our modified CSIsat tool for that particular counterexample. Here let the location number 1 correspond to the variable `s_1` and similarly for the location number 2 and the variable `s_2`, let the list structure have two fields (`f_1` and `next`) corresponding to the numbers 0 and 1, and finally let the constant N be equal to 1000. The path formula of the counterexample is shown in figure 1.

The interpolant produced by our modified CSIsat tool is

$$V_6\Big(S\big(L(V_6(2)),0\big)\Big) = 2 \vee S\big(L(V_6(2)),0\big) < 1000$$

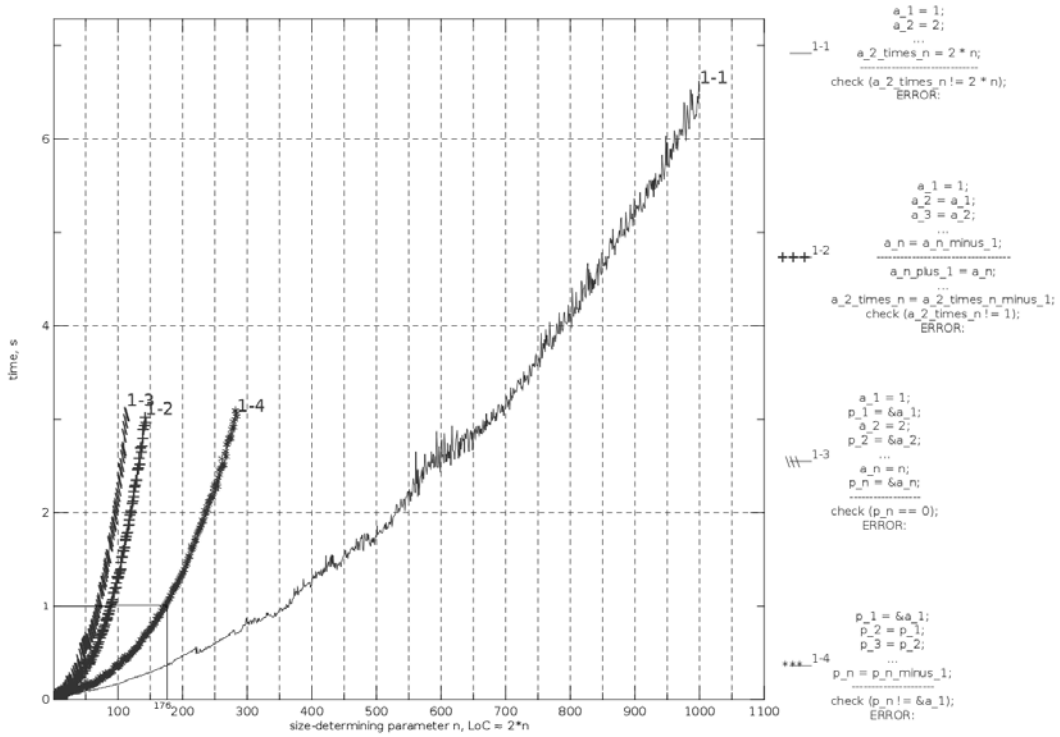
that corresponds to the predicate

$$V\Big(S\big(L(V(2)),0\big)\Big) = 2$$

that is `s_2->f_1 == 2`. The condition is indeed met in the specified program location and proves the infeasibility of the counterexample. The same predicate can be used to prove the infeasibility of another error path through the same program location, e.g.:

```
s_1->f_1 = 1;
s_1 = s_1->next;
s_1->f_1 = 2;
s_2 = s_1;
s_1 = s_1->next;
```

Fig. 2. Results on specially generated benchmarks (simplest cases).




```
----------------------
check(s_2->f_1 == 2); // 'else' branch
                      // in the same 'if'
                      // statement
s_1->f_1 = s_2->f_1 - 2;
check(s_1->f_1 != 0);
ERROR:
```

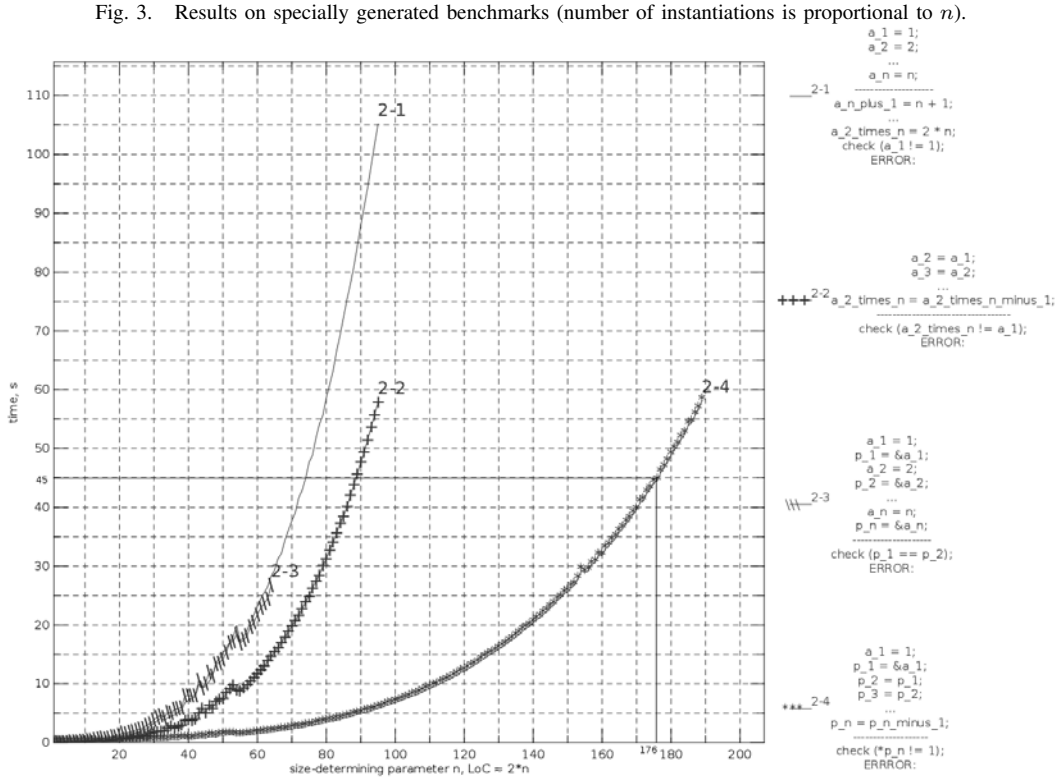
The results of benchmarking our tool on the specially generated example set are shown in figures 2 and 3. The legends of the plots contain code snippets corresponding to the path formulae used in each of the examples. The parameter n specifies the sizes of the test cases so that they contain approximately $2n$ lines of code. The cut-point location dividing the counterexample path into parts is marked with a dashed line and the `check` operator designates chosen branching condition the same way as in the previously considered example. The label `ERROR` is unreachable in the examples so the corresponding path formulae are unsatisfiable. They are built the using the uninterpreted functions presented above and are passed to the modified CSIsat tool with options `-extrInsts` and `-simplInsts` (the options enable our instantiation extraction and simplification heuristics in CVC3). All the benchmarks were launched with the time limit of 600 seconds and memory limit of 1 GiB. The launching for each of the examples was aborted after five abrupt terminations of the tool (including time, memory limit exceeding and uncaught exceptions).

Figure 2 presents the results for very simple cases without manipulating any structures or arrays. They also require no quantifier instantiations of memory update expressions (each variable value is used just after the assignment). Figure 3 presents the results for the very similar test cases. But these ones require the number of quantifier instantiations that is linear in proportion to the number of lines of code in the example. The plots show exponential growth of interpolation time in all the tests. But the tests requiring many quantifier instantiations took up to 45 times and even more greater amount of time for interpolation in comparison with the similar tests requiring very few instantiations (see the plots in figures 2 and 3 for $n \approx 176$).

For the benchmarking results on less trivial cases see the figures in the complete version of this paper ([21]).

Overall the results have shown that our modified tool can produce interpolants for quantified formulae with relatively small number of quantifier instantiations. But the size of real-life counterexamples, e.g. in verifying Linux kernel drivers, is about 1000 to 10000 lines of code and the number of interpolator calls is about 10 to 30 per driver (see the paper [17] for details). This supposes that to achieve a suitable verification time of about 15 minutes per a driver, the tool should produce interpolants for the formulae for about 5000 LoC in about 45 seconds. As the presented results have shown, the tool can process formulae for not more than about 350 (176×2) LoC in that time even in the very simple cases (see figure 3). The modified tool thus turned out to be not efficient enough to be used in a scalable model checker implementing

Fig. 3. Results on specially generated benchmarks (number of instantiations is proportional to n).



the logical memory model considered. This inefficiency can be explained with relative inefficiency of CSIsat internal decision procedure as well as the use of some heuristics for necessary instantiation extraction. The heuristics rather frequently extract unnecessary instantiations. They sometimes also fail to extract the necessary ones causing the tool to terminate abruptly.

VI. Conclusions

In this paper we proposed a relatively easy approach to implement an instantiation-based interpolating decision procedure for quantified formulae based on an existing interpolating solver and a modern SMT solver with quantifier support. We have implemented our approach in the modified version of the CSIsat interpolation tool using CVC3 as an external SMT solver. The modified tool implements Craig interpolation for quantified formulae with respect to the theories of linear real arithmetic and uninterpreted functions with equality.

We have also performed a preliminary benchmarking of our modified tool on two distinct benchmark sets. The first was obtained form the SMT-LIB benchmark set while the second contained the specially generated benchmarks simulating the interpolation tasks that a real model checker could give to our tool in case it implemented a location-based logical memory model for the discovery of pointer predicates.

The tool benchmarking results on our both test sets have shown that our proposed approach is generally functional, but lacks enough efficiency to be used for predicate derivation in a real industrial model checker.

The CSIsat internal decision procedure used in the tool to derive the unsatisfiability proof tree for interpolation is a way less efficient than that of CVC3 and other state-of-the-art SMT solvers. Hence one of the interesting further research areas here is more comprehensive analysis of CVC3 unsatisfiability proof trees in order to use them for reducing the overhead of proving the formula unsatisfiable once again in the interpolator itself. This way we can avoid the most significant burden bounding the efficiency of our tool.

References

[1] Thomas Ball, Ella Bounimova, Rahul Kumar, and Vladimir Levin. SLAM2: Static driver verification with under 4% false alarms. In *Formal Methods in Computer Aided Design*, 2010.

[2] Thomas Ball, Ella Bounimova, Vladimir Levin, and Leonardo De Moura. Efficient evaluation of pointer predicates with z3 smt solver in slam2. Technical report, 2010.

[3] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[4] Clark Barrett, Aaron Stump, Cesare Tinelli, Sascha Boehme, David Cok, David Deharbe, Bruno Dutertre, Pascal Fontaine, Vijay Ganesh, Alberto Griggio, Jim Grundy, Paul Jackson, Albert Oliveras, Sava Krstić, Michal Moskal, Leonardo De Moura, Roberto Sebastiani, To David Cok, and Jochen Hoenicke. C.: The smt-lib standard: Version 2.0. Technical report, 2010.

[5] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.

[6] D. Beyer, T.A. Henzinger, and G. Théoduloz. Lazy shape analysis. *Proc. CAV, LNCS*, 4144:532–546, 2006.

[7] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. The software model checker Blast: Applications to software engineering. *Int. J. Softw. Tools Technol. Transf.*, 9(5):505–525, 2007.

[8] Dirk Beyer and M. Erkan Keremoglu. CPAchecker: A tool for configurable software verification. Technical report, School of Computing Science, Simon Fraser University, 2009.

[9] Dirk Beyer, Damien Zufferey, and Rupak Majumdar. Csisat: Interpolation for la+euf. In *CAV*, pages 304–308, 2008.

[10] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The mathsat 4 smt solver. In *CAV*, pages 299–303, 2008.

[11] Juergen Christ and Jochen Hoenicke. Instantiation-based interpolation for quantified formulae. In Nikolaj Bjorner, Robert Nieuwenhuis, Helmut Veith, and Andrei Voronkov, editors, *Decision Procedures in Software, Hardware and Bioware*, number 10161 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[12] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Emerson and A. Sistla, editors, *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer Berlin / Heidelberg, 2000. 10.1007/10722167_15.

[13] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. *Tools and Algorithms for the Construction and Analysis of Systems*, 4963:337–340, 2008.

[14] Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with pvs. In *Computer Aided Verification, 9th International Conference, CAV '97, Haifa, Israel*, pages 72–83, 1997.

[15] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Kenneth L. McMillan. Abstractions from proofs. *SIGPLAN Not.*, 39(1):232–244, 2004.

[16] Ranjit Jhala and Rupak Majumdar. Software model checking. *ACM Comput. Surv.*, 41(4):21:1–21:54, October 2009.

[17] Alexey Khoroshilov, Vadim Mutilin, Eugene Novikov, Pavel Shved, , and Alexander Strakh. Towards an open framework for C verification tools benchmarking. In *Proceedings of PSI*, pages 82–91, Akademgorodok, Novosibirsk, Russia, 2011.

[18] Kenneth L. McMillan. An interpolating theorem prover. In *TACAS*, pages 16–30, 2004.

[19] Kenneth L. McMillan. Interpolants from z3 proofs. Technical report, Microsoft Research, 2011.

[20] Leonardo Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. In *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction*, CADE-21, pages 183–198, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] Vadim Mutilin and Mikhail Mandrykin. Instantiation-based interpolation for quantified formulae in csisat. In *Proceedings of the Institute for System Programming of the Russian Academy of Sciences (in Russian)*, 2012.

[22] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.

[23] Pavel Shved, Vadim Mutilin, and Mikhail Mandrykin. Static verfication "under the hood": Implementation details and improvements of BLAST. In *Proceedings of SYRCoSE*, volume 1, pages 54–60, 2011.

Translation of UML Statecharts to UPPAAL Automata for Verification of Real-time Systems

Daniil A. Zorin
Department of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
juan@lvk.cs.msu.su

Vladislav V. Podymov
Department of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
vvpodymov@gmail.com

Abstract — **In this paper we present a tool to transform UML statecharts to UPPAAL automata. The tool allows one to check temporal properties against statecharts modeling a real-time system. We give the constraints on statecharts, the tool description, and the results of testing it on a well-known traffic control example.**

Keywords — verification, UML, UPPAAL, modeling, real-time systems

I. INTRODUCTION

Usually verification tools work with models written in specialized languages intended for convenient application of verification algorithms. On the other hand, during the design stage systems are often modeled with universal modeling languages (such as UML) or industry-specific modeling languages. UML statechart diagrams are an example of universal models describing the behavior of systems communicating with the environment via shared memory and message passing. Real-time systems are often modeled with such diagrams. Since the cost of correcting an error increases over the course of system development, verifying the properties of the system as early as possible one improves its quality and simplifies its development.

In this paper we present a tool for converting UML statechart diagrams to timed automata used in the UPPAAL verification system [1, 2]. In section 2 we define the syntax of expressions we use in UML diagrams. The algorithm is discussed in section 3. Experimental results obtained with the algorithm are given in section 4.

II. UML STATECHARTS

Unified Modeling Language (UML) is used to design a wide range of systems implemented in various languages and in different environments. Therefore, the authors of the standard of UML deliberately avoid defining syntax and semantic of the language completely [5, ch. 13]. The language defines a metamodel comprised of syntactical constraints on all models in UML notation. Generally it is only possible to say whether the model is syntactically correct. The behavior of a correct model might be undetermined in some cases: guards, actions and triggers can be defined in a natural language which tolerates different interpretations.

The authors of the language suggest creating a separate profile for every class of systems without changing the general notation. However, in the case of statechart diagrams, creating the profile does not solve interpretation problems. To prove the properties formally it is necessary to define a strict syntax and semantic of all used primitives of statecharts. In this study, additional constraints on the structure of the diagrams and the syntax of expressions are imposed, thus the ambiguity is avoided.

Simple states are the same as in the standard UML metamodel. There are two types of composite states: sequential and parallel. Automata residing in a parallel state are executed simultaneously. Composite states have special entry and exit states.

Some states are marked with logical formulae called invariants; a system can reside in such state only while its invariant is satisfied.

Each transition between states may be provided with a guard, an action, and a synchronization. Guards express requirements that must be satisfied to enable the transition. Actions are the operations performed after the transition is fired.

The syntax of guards, invariants and actions is similar to the syntax of the C language. There are three types of variables: an integer type over a certain range (e.g., int [4..9] x = 5;), the boolean type (e.g., bool b = false;), and the clock type (e.g., clock c;). All variables must be defined in the comments section of the UML model. Expressions admitted in guards include all types of comparison as well as logical NOT, AND and OR operations. Actions may contain assignment statements including complex arithmetic expressions and the C-style ternary operator '? :'. Invariants have the same syntax as guards do, though the expression must be marked with the keyword 'assume()'.

There are two additional expressions in the syntax. The boolean expression 'in(S)' borrowed from STATEMATE language [3] denotes that the state S is active in the system. The operation of random assignment, written as 'x=random();' non-deterministically gives a value to an integer or boolean variable admitted by the type.

The syntax and the meaning of macros are similar to the ones in C language. They are defined in the comments section

along with the variables. The macro '#define X Y' replaces all occurrences of X with Y before other stages of the translation.

The examples of expressions can be found in the Figure 9.

The operation of sending a signal is identical to the hardware-like message broadcast [3]. Every signal must be defined in the model. When signal S is sent by a transition (denoted by the synchronization section), the automaton marks signal S as sent, and on the next step all the automata that can activate a transition with the receiving of signal S (written as '!!S') must do that. If none of the automata can receive the signal, it is considered lost. For instance, on figure 10 the system moves from state AHome to state AToGreen only at receiving a signal AtoG.

III. TRANSLATION OF UML STATECHARTS TO UPPAAL AUTOMATA

The UML to UPPAAL translator works with UML statecharts in the widespread XMI format. When a file is parsed and an internal representation is constructed, the translation is performed in two phases. First, the statechart is transformed to the intermediate form – an hierarchical timed automaton (HTA) [4] – and then this automaton is translated to a network of timed automata (NTA) according to the algorithm similar to the one introduced in [4].

Since the structure of statecharts differs significantly from the structure of hierarchical timed automata, an additional step of transformation of UML statecharts should be carried out before translating them to UPPAAL.

Firstly, during the parsing of UML, the expressions that do not belong to UPPAAL model language are translated. All macro substitutions take place before parsing the guards and actions. The 'in(S)' expression in guards is replaced by checking the value of a special flag variable which is unique for each 'in(S)' statement.

Further, all references to automata are replaced by their unique copies. If one automaton is nested into another one, it is inserted as well. Name collision on this step is avoided: if the names of two states in two nested automata coincide, then one of the states is renamed, and if two variables with the same name are declared in different scopes (e.g. in two automata referenced in the third one), then one of them is renamed. As a result a single hierarchical UML statechart is formed.

The next step is to modify composite states (figure 2-3). In HTA, only transitions between simple states, entry and exit states are allowed, so it is necessary to change the arcs which start or end in composite states to match them with the corresponding entries and exits. Adding several new entries or exits might be necessary. In HTA transitions into a composite state are allowed if they end in its entry state, similarly, transition out of a composite state into its parent is possible if it starts in an exit state. All other transitions must begin and end inside of the same composite state, i.e., the source and target states remain in the same composite state. However, in UML statecharts it is possible to perform transitions to a deeply nested state; hence it is important to add all exits and entries in between.

Finally, guards, actions, and synchronizations should not be present on transitions ending in exit states according to HTA definition. In such cases, a new state, like tmp in the Figure 2, is added and the guards, actions and synchronizations are assigned to the transition ending in the new state.

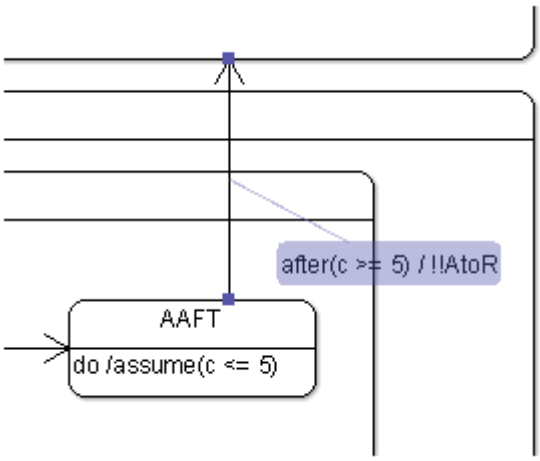


Figure 1. Correction of composite states: before

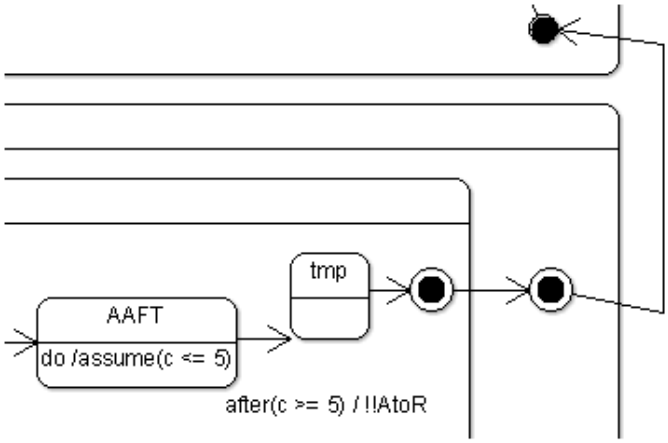


Figure 2. Correction of composite states: after

When HTA is obtained, it is translated into NTA used in UPPAAL.

NTA consists of processes, variables, channels and clocks. A process is a certain timed automata which has finite sets of locations and transitions.

Some locations are marked with invariants, and some transitions are supplied with guards, actions, and synchronizations. Invariants, guards, actions, and synchronizations are similar to those in HTA.

Three kinds of locations are possible: ordinary, urgent, and committed. When an urgent location is active in NTA, no time can advance, and if the location can be deactivated, it is left at once. Committed locations are similar to urgent locations, but they have the highest priority in deactivation.

Each channel has its own type, either broadcast or handshake. Broadcast channels are similar to those in HTA. Handshake channel is used to synchronize the execution of exactly two transitions in NTA.

The translation HTA to NTA is as follows.

Before state translation, variables, channels and clocks are copied from HTA directly to NTA. According to translation algorithm, auxiliary variables and channels are added. Some of them are mentioned below.

Every composite state S in HTA corresponds to a process P(S) in NTA. Every such a process has an initial location ‘idle’ which corresponds to inactivity of a composite state.

Consider a parallel composite state S in HTA. A special location ‘active’ is created in P(S). The ‘active’ location can be reached from the ‘idle’ location by performing a sequence of transitions via committed locations ‘start(X)’, one for each composite state X nested in S. The first transition in the sequence carries a synchronization ‘activate(S)?’ that activates P(S). Other transitions in the sequence carry synchronizations ‘activate(X)!’ for every nested state X. Also there is exactly one transition from the state ‘active’ to the state ‘idle’ that carries a synchronization ‘deactivate(S)?’ deactivating P(S).

When P(S) is activated, the whole sequence of transitions is executed with no time advancing, every nested state is activated, and then P(S) reaches the ‘active’ location which corresponds to activity of all states nested in S.

Consider a sequential composite state S in HTA. A process P(S) includes locations ‘active(X)’ for every state X nested in S as well as committed locations ‘start(X)’ for every composite state nested in S. Locations ‘start(X)’ and ‘active(X)’ are connected via a transition decorated with a synchronization ‘activate(X)!’. The ‘idle’ location is connected with either a location ‘start(X)’ in the case of a composite state X or with a location ‘active(X)’ in the case of a basic state X via transition with synchronization ‘activate(S)?’.

When P(S) is activated, it activates exactly one of its nested states and reaches one of ‘active(X)’ locations which correspond to the activity of X.

To deactivate a state X nested in S, the process P(S) uses a set of deactivation sequences of committed locations. Transitions in each sequence carry synchronizations ‘deactivate(Y)!’ for every composite state Y nested at any level in X. Thereby when a deactivation sequence is executed, all inner states which can be deactivated simultaneously in HTA are deactivated in NTA. If S has to be deactivated as well, the final location of the sequence is connected to the ‘idle’ location. Otherwise it is connected to one of ‘start(X)’ or ‘active(X)’ locations.

To initialize the NTA defined above, an additional process ‘Kickoff’ is created. This process is a sequence of committed locations which ends with an ordinary location. Transitions in this process carry special synchronizations ‘init(X)!’ for every initial state X of HTA. Special initial transitions are also added into other processes to reach a correct initial state.

IV. Experimental Results

To be certain that the implementation of our translation algorithm is correct and well suited for composition with UPPAAL we tested it on several case studies. The simple examples were used to make sure that the output of the algorithm satisfies the expectations and to check the behavior on various sample cases. Some more complex tests were aimed to simulate the whole process of verification of a system defined by a UML statechart diagram. Below we present the results of our experiments with the model of traffic lights control system described in [4].

A. *Simple tests*

An example of a simple test is given on figures 4-5.

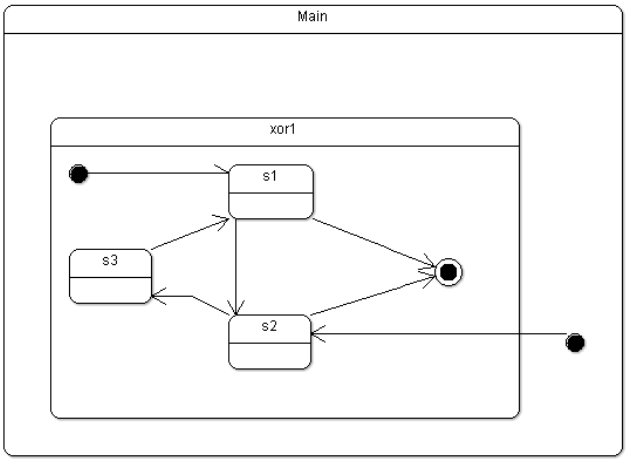


Figure 3. Example 1: UML

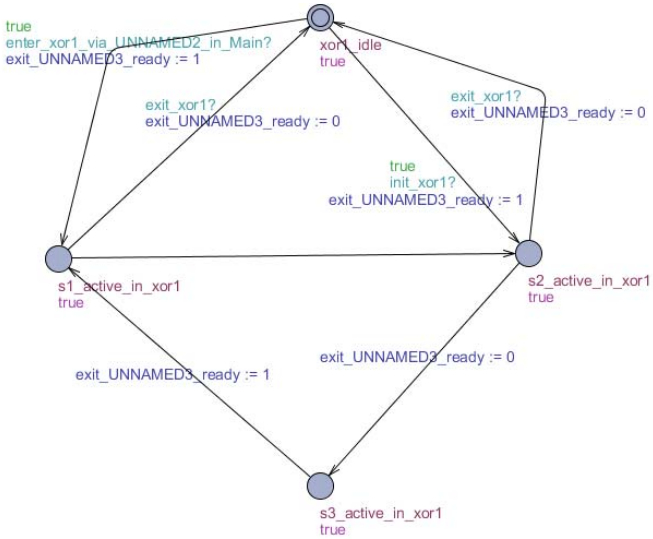


Figure 4. Example 1: UPPAAL

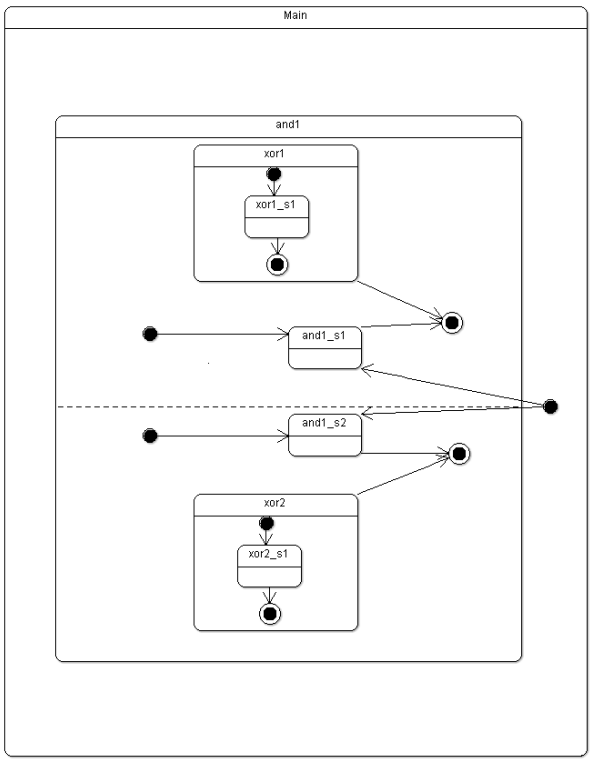


Figure 5. Example 2: UML

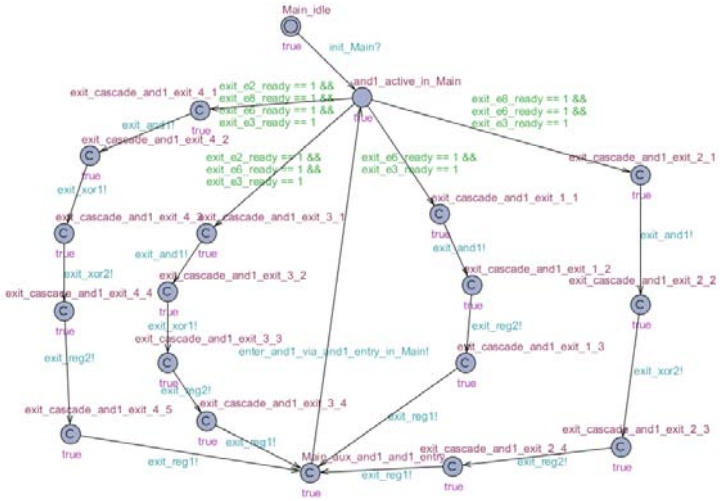


Figure 6. Example 2: UPPAAL

B. *Traffic lights example*

The traffic lights control system consists of two traffic lights on a crossroad. The lights are controlled by a processor supplied with some sensors. Lights on the street and on the avenue change colors customary to let cars pass by in both directions. Further, in the case an ambulance car arrives from any direction, the lights must turn to green on that direction in order to let the ambulance pass as soon as possible.

The UML diagrams for this system are shown in the Figures 10-11. The first diagram contains state loops for the lights and the ambulance and a reference to the diagram of the light controller. The lights are changed in the usual order (green to yellow to red) according to the signals of the light controller. The ambulance appears non-deterministically and passes through the street crossing.

The light controller normally sends signals to the lights to switch their colors in order. When the ambulance appears, the system exits the normal cycle and enters the AmbulanceArriving composite state where the light colors are changed arbitrarily in order to turn the light on the street where the ambulance is waiting green.

In [4] the authors constructed a UPPAAL model for this system manually to verify its properties. We used our translator and obtained the model automatically.

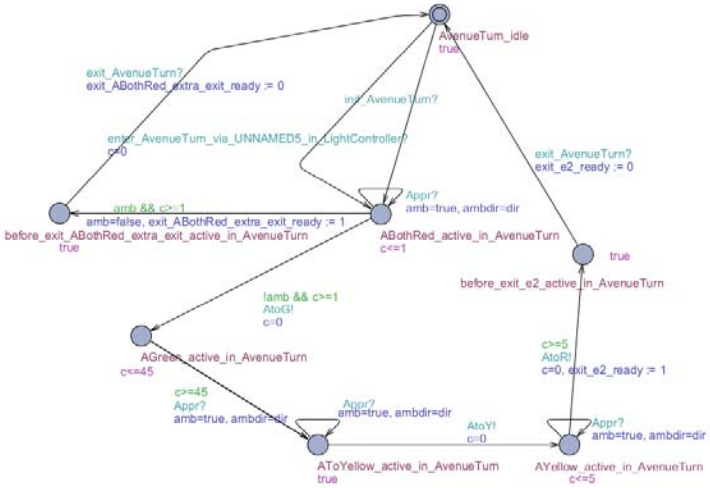


Figure 7. UPPAAL diagram for AvenueTurn composite state

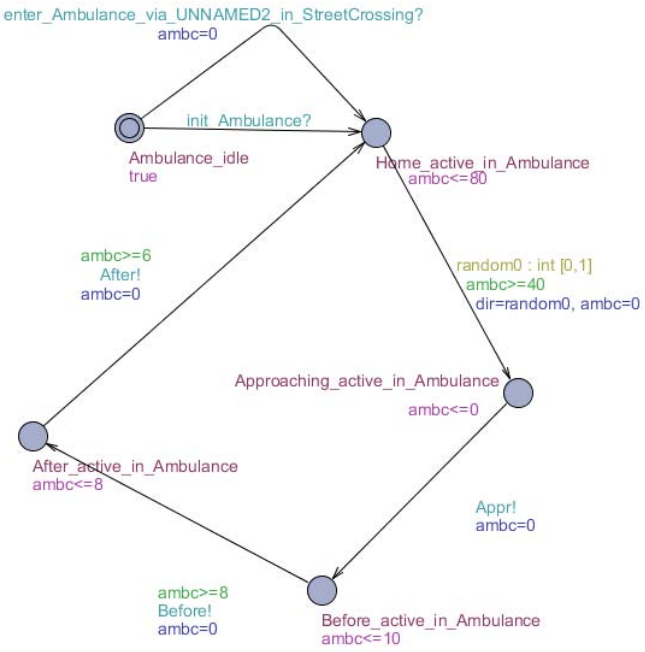


Figure 8. UPPAAL diagram for Ambulance behavior

Some of the UPPAAL automata are shown on figures 7-8.

The following properties were tested.

A[]! deadlock

This property guarantees the absence of deadlocks.

A[]! (stg==1 || sty==1) imply avr==1

A[]! (avg==1 || avy==1) imply str==1

The lights are synchronized: if the avenue light is green or yellow, the street light must be red and vice versa.

E<> stg==1 && avg==1

This property means that there exists a trace where both lights are green at the same time and it was proved to be false. At the same time the seemingly contrary property

A[] (stg==1 || avg==1)

is not fulfilled also, because there can be a situation where one light is red and the other one is yellow.

Ambulance_process_proc.Approaching_active_in_Ambula nce -->

Ambulance_process_proc.Home_active_in_Ambulance

Home state for the ambulance car is reachable from the Approaching state, which basically means that the ambulance will always eventually pass the crossing.

CONCLUSIONS

Experiments with our tool testify that translation of UML statecharts to UPPAAL timed automata is possible. We reproduced the results that were obtained manually in [4] with our automatic translation and showed that the tool is applicable to models of relatively simple real-time systems with parallel interacting processes. Further work includes formal proof of the correctness of the algorithm based on [3] and testing the tool on practical examples of real-time systems.

REFERENCES

[1] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. Int. Journal on Software Tools for Technology Transfer, 1(1–2):134–152, October 1997.

[2] Alexandre David, Gerd Behrmann, Kim G. Larsen, and Wang Yi. A tool architecture for the next generation of UPPAAL. In 10th Anniversary Colloquium. Formal Methods at the Cross Roads: From Panacea to Foundational Support, LNCS, 2003.

[3] David, M. Moller Oliver, Wang Yi. Verification of UML Statechart with Real-Time Extensions / Uppsala: Department of Information Technology, Uppsala University. IT Technical Report 2003-009, 2003.

[4] A. Furfaro, L. Nigro. A development methodology for embedded systems based on RT-DEVS, Innovations in Systems and Software Engineering, vol 5, P. 117-127, June 2009.

[5] Grady Booch, Ivar Jacobson & Jim Rumbaugh. OMG Unified Modeling Language Specification. Addison Wesley, 1997

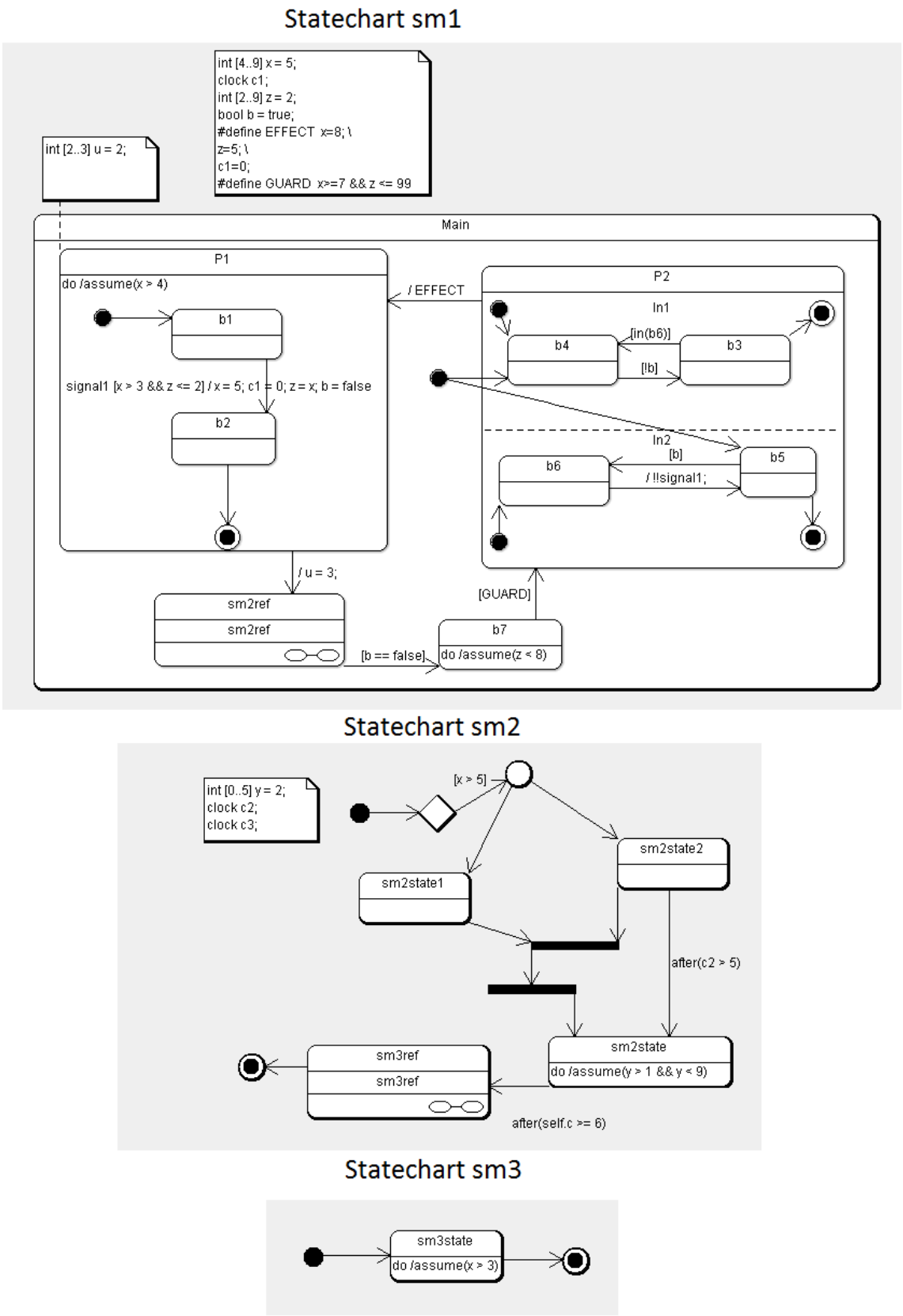


Figure 9. Example of a UML diagram containing all syntactic features

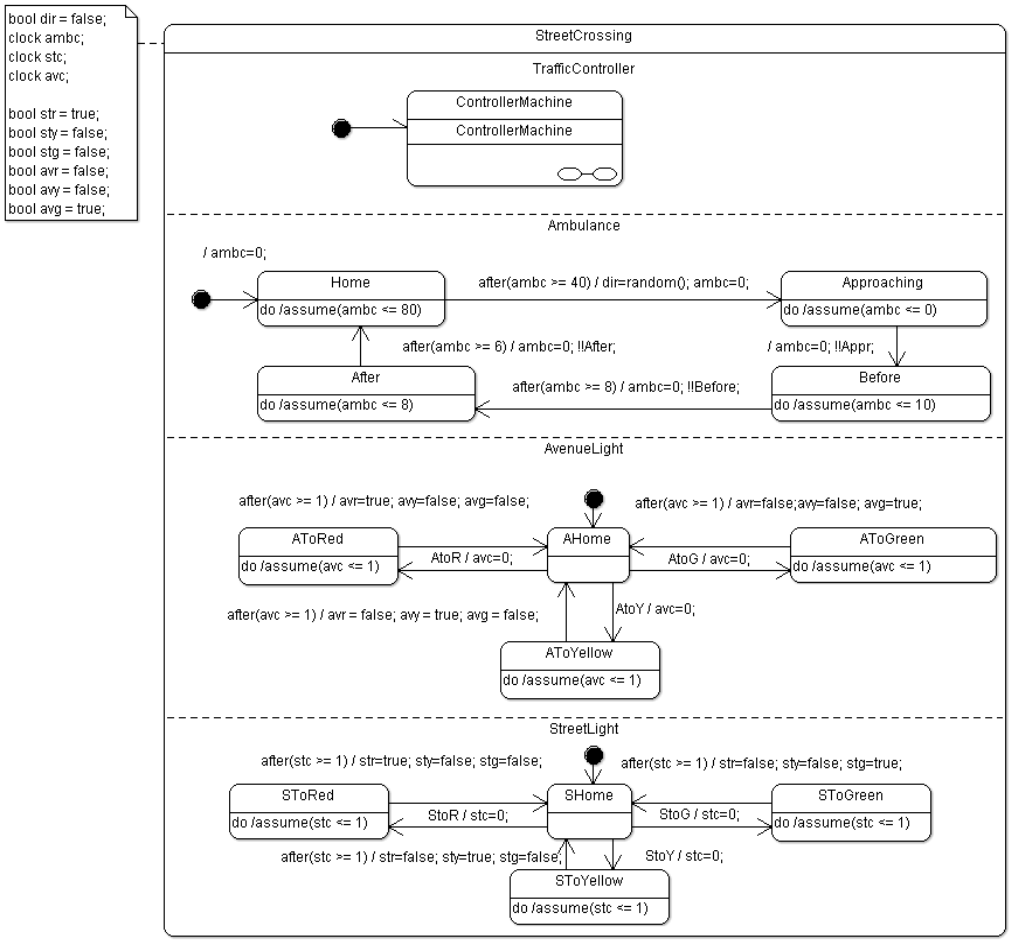


Figure 10. Traffic lights UML diagram

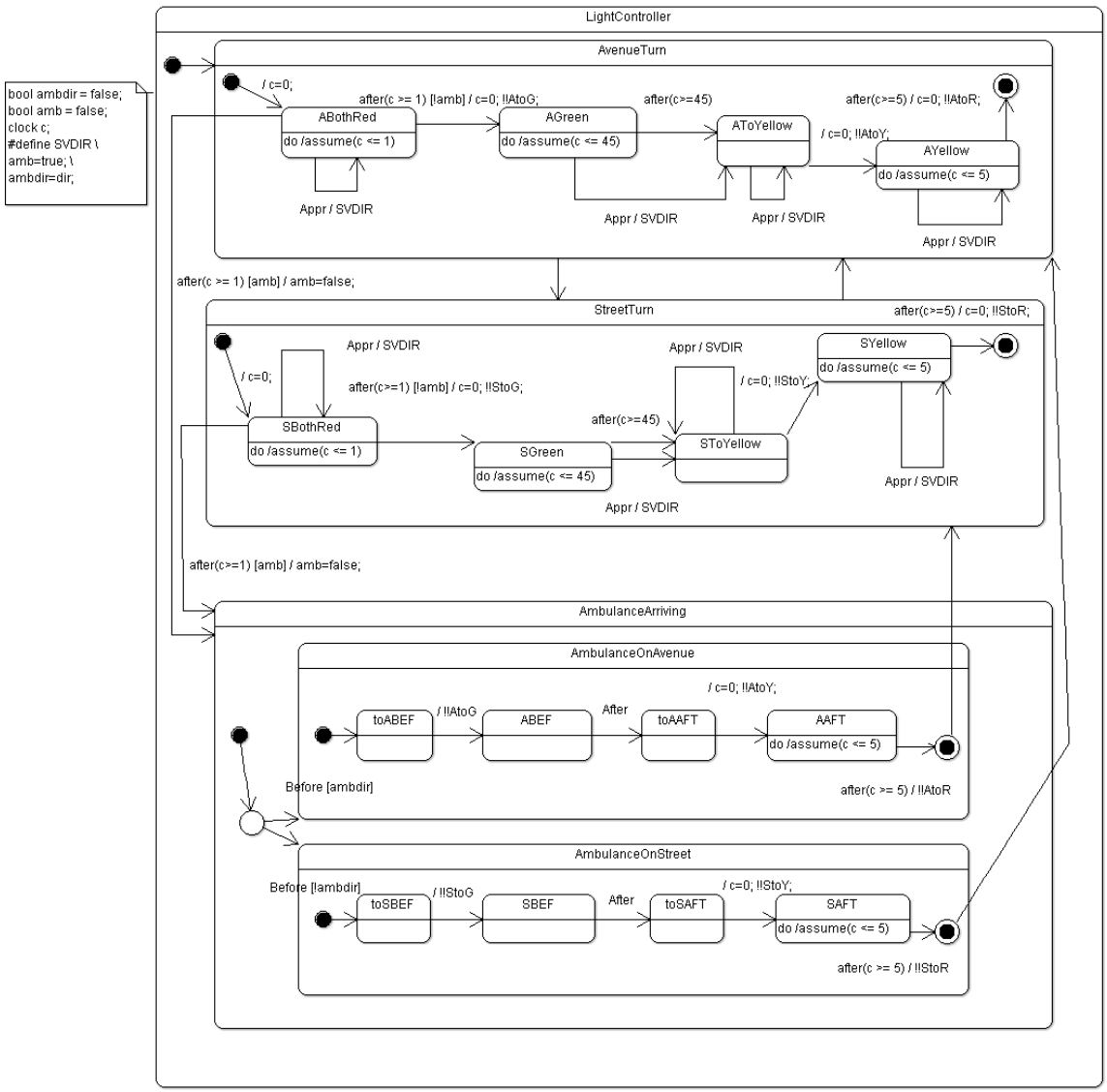


Figure 11. Traffic lights UML diagram (2)

Enhancement of automated static verification efficiency through manual quantifiers instantiation

Denis Buzdalov[1]
Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russia
Email: buzdalov@ispras.ru

***Abstract*—Checking of a program conformity with a contract specification is a hard problem. Usually this check requires high time and memory expenses. This work describes a technique that allows to lower verification costs through usage of some information known by people who verify a program and contains suggestions of the way how to organize it.**

I. Introduction

Nowadays a lot of computer systems perform quality critical tasks. It means that bad quality of work of these systems can lead people's deaths and injuries or financial losses. That's why such systems need to be *verified*.

Verification requires checked properties and possibly some additional info to be specified in some formal language. Such collection of formal statements is called a *formal specification*. Formal specification has precisely defined semantics which is used by a verification instrument.

There is a lot of ways of verification. They vary on resources requirements and quality of a result. *Formal static verification* differs from e.g. testing in confidence of specification conformity in case of positive verification verdict. Unlike the model checking this approach can be used for verification of a target system itself but not its model. Also, this approach can be used for a wide class of target systems (*e.g. for checking of embedded systems*).

There are many types of specifications. *Contract specifications* [1] is a popular type which has a lot of supporting instruments. Contract specification is a set of statements of the following types:

- *precondition* is a condition which holding is required for an operation (function, method, subroutine) execution; *e.g. a function of real square root computation will have a non-negativity of its argument as a precondition*;
- *postcondition* is a condition which have to be held at the end of an operation execution; *the square root function can have a condition that the result multiplied by itself must be equal to the input argument as a postcondition*;
- *invariant* is a condition that have to be held at some time or on some events arising

Instruments of contract-based static verification reduce the program correctness proving task to the task of *satisfiability* of some *typed predicates* (these predicates can contain some operations and relations of some typed values).

Satisfiability is a laborious task. This task is usually not solved manually because of usually big number of predicates which satisfiability have to be proved. Also manual proofs are not stable to changes of program, so it cannot be used during the program development. That's why automatic or automated *solvers* are used for such tasks (e.g. SMT-solvers [2]).

II. Motivation

Specification itself is usually not enough for successful verification of a correct program. For example, Floyd methods [3] and Hoare rules [4] expect each loop to be marked by a loop invariant (a predicate which is hold before each loop condition check). Ability and time of proving highly depends on which loop invariants are chosen.

But even if every loop has an invariant allowing to prove a program correctness it may be not enough for the successful verification.

A person that wants to prove a program correctness has to add *lemmata* and *assertions* to a specification to help solver. These additions may reflect different properties of data and its operations which solver it not able to understand itself. *For instance, if we have a* $f : List \to Multiset$ *operation and* $List$ *and* $Multiset$ *types have an addition operation (concatenation and union correspondingly), then lemma of linearity of* f *relatively to the addition operation will have form of* $\forall l1, l2 : List \cdot f(l1 + l2) = f(l1) + f(l2)$

These lemmata are conjuncted with a precondition during proving of a fact that a precondition implies a postcondition ($p_{pre} \wedge p_{lemma1} \implies p_{post}$). This allows lemmata statements to be used in the verification process.

Assertions are similar to lemmata but unlike them are defined inside operations and hold only inside them. Consequently, assertions can represent properties connected with local data and also consider a precondition to be held (be implied by it). Assertions also are conjuncted with a precondition for the verification process.

There are some instruments that support the described correctness proving technique and can be used for the formal static verification of software e.g. frama-c [5] and Dafny [6].

Lemmata and assertions often are statements with the universal quantifier.

[1]This work is partially supported by RFBR 11-07-12075-ofi-m, 10-07-1047a, Minorbnauki RF 07.514.11.410.

One of the main difficulty of the satisfiability problem solving is a successful usage of such statements. The way how to instantiate an expression of the quantifier is not defined by algorithms that are used in solvers [7]. But effectivity of satisfiability proving highly depends on how instantiation is performed. There are some heuristic methods of instantiation that increase the proving speed [8], [9] but they give positive results not often in the practice. That's why verification usually requires high resources amount (both time and memory).

There can be a lot of lemmata but not all of them are required for a check of each postcondition. But still, solver will try to instantiate useless statements with the universal quantifier. This can considerably increase proving costs.

This work offers a technique that makes verification easier by reducing of quantified statements usage in lemmata and assertions and also by limiting of usage of useless lemmata.

III. Suggested technique

Technique is based on the fact that a person who is trying to verify a program unlike modern solvers usually knows namely which lemmata help or can help for the postcondition or invariant checking. Also he usually understands how a statement with the universal quantifier should be instantiated to use lemma properly.

Consider a lemma of linearity of a f function from the example above. To prove that removing of the first element from a list leads a multiset of its elements to decrease its size by one, it is enough to take a sublist containing only the first element of the original list as $l1$ and a tail of the list as $l2$.

To use such knowledge it is offered to rewrite lemmata that are formed like

$$\forall x_1 : X_1, x_2 : X_2, \ldots, x_n : X_n \cdot$$

$$\cdot A(x_1, x_2, \ldots, x_n) \implies P(x_1, x_2, \ldots, x_n)$$

as *pure functions* of form $t : X_1 \times X_2 \times \ldots \times X_n \to \varnothing$ and having the $A(x_1, x_2, \ldots, x_n)$ predicate as a precondition and the $P(x_1, x_2, \ldots, x_n)$ predicate as a postcondition. Pre- and postconditions have the same set of parameters because of a function purity and the fact that it does not return any value. Let's call functions of the described type that represent some lemma a *function-lemma*.

Such lemmata representation allows to move statements that help to prove a lemma but useless for proving other lemmata, inside the function-lemma. That makes solver's task easier because it reduces count of statements that it can but shouldn't use. That allows to reduce verification costs.

Besides, this representation allows to use once proved lemma without rewriting a proof. *There's a lack of such ability in existing instruments, e.g. in frama-c [5]. This fact makes lemmata usage difficult and increases a solver's work amount.*

But still, if the lemmata meaning isn't changed, namely that all lemmata can be used for the verification, the problem of instantiation of parameters (the same problem with the problem of instantiation of quantifier variables) remains. To solve this problem it is suggested to use lemmata only in places pointed by a person that verifies a program (except situation mentioned below).

When some lemma is pointed to be used, parameters of the function-lemma have to be explicitly defined. At the place of pointing applicability of the lemma should be checked (through the function-lemma's precondition check) and if the check is successful the main lemma statement (represented by the function-lemma postcondition) have to be considered to be held (because the lemma is proved separately).

Such lemmata usage can be both standalone and inside a composite statement.

No statements with the universal quantifiers appear where they are not needed if such lemmata semantics is used. If statements with the universal quantifier are essential, then instantiation variants count is not increased comparing to the existing lemmata semantics. Moreover, verificating person can considerably decrease this count if he thinks that it is enough for proving. Consequently verification resources requirements are decreased if lemmata are used properly.

However usage of the described semantics can reduce proving quality (compared to the existing semantics) when lemmata are used improperly. To make this semantics to be not worse that the existing one, it is modified.

If proof that uses pointed by a man function-lemmata is unsuccessful, then statements of function-lemmata can be interpreted as statements with the universal quantifier and after that proving should be continued. So if a program correctness can be proved without usage of the suggested technique, then it can be done with the suggested technique too. In that case difference of verification resources expenses will be not big because the suggested technique does not increase instantiation variants count (and often decreases it up to the single one).

Right function-lemma usage can considerably decrease the verification expenses of a correct program because it eliminates usage of statements with the universal quantifier and usage of useless lemmata.

Moreover this technique allows to use named properties in a convenient way. This allows to standardize and describe lots of widely used data and operation properties.

For instance, consider a lemma of the square operation $sqr : real \to real$ being converse to the square root operation $sqrt : real \to real$. Let the lemma be named $sqrToSqrtConversity$. It has a single parameter, let it be named x. The precondition will be $x \geqslant 0$, and the postcondition will be $sqr(sqrt(x)) = x$.

Consider an operation that has an argument a and property of sqr being converse to $sqrt$ have to be held for this argument. Then the $sqrToSqrtConversity(a)$ statement should be added as a precondition. If some operation returns a numbers set s which all are numbers that the converse property have to be applicable to, then if is enough to write $\forall x \in s \cdot sqrToSqrtConversity(x)$ as a postcondition.

Consider a function evaluating $(\sqrt{x}+\sqrt{y})^2$. Then to prove a property of "$y = 0$ implies the result to be equal to x" we can use the $sqrToSqrtConversity$ lemma with the x number as a parameter (also the $sqrt(0) = 0$ property will

be required).

IV. Case study

Considering existing instruments, Dafny [6], a static verification instrument, partially supports the suggested technique. It supports an ability of defining of functions that can be used as a function-lemma. However these functions cannot be used inside composite statements. That fact does not allow to use some technique abilities.

These limited abilities were, however, enough to prove permutation correctness of the Shell sort algorithm [10] with Sedgewick coefficients [11] using acceptable time (approx. 10 min.) and memory (approx. 300 MB) amount. Author hasn't managed to do this without the suggested technique using up to 3 hours and 4 GB of memory.

V. Conclusion

A technique that widens ability of automatic verification instruments (on appropriate instruments modernization) was suggested.

This technique was used in practice, as far as it is possible using existing instruments. The technique allowed to prove a program correctness which couldn't be done without usage of this technique.

References

[1] B. Meyer, "Design by contract," Interactive Software Engineering Inc., Tech. Rep. TR-EI-12/CO, 1986.

[2] C. Barrett, A. Stump, and C. Tinelli, "The satisfiability modulo theories library (SMT-LIB)," http://www.smtlib.org/, 2010.

[3] R. W. Floyd, "Assigning meaning to programs," in *In Proceedings of Symposium on Applied Mathematics*, vol. 19. A.M.S., 1967, pp. 19–32.

[4] C. A. R. Hoare, "An axiomatic basis for computer programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969.

[5] "Frama-c," http://frama-c.com/.

[6] "Dafny," http://research.microsoft.com/en-us/projects/dafny/.

[7] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.

[8] Y. Ge and L. Moura, "Complete instantiation for quantified formulas in satisfiabiliby modulo theories," in *Proceedings of the 21st International Conference on Computer Aided Verification*, ser. CAV '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 306–320.

[9] L. Moura and N. Bjørner, "Efficient e-matching for smt solvers," in *Proceedings of the 21st international conference on Automated Deduction: Automated Deduction*, ser. CADE-21. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 183–198.

[10] D. L. Shell, "A high-speed sorting procedure," *Commun. ACM*, vol. 2, no. 7, pp. 30–32, Jul. 1959.

[11] R. Sedgewick, "A new upper bound for shellsort," *Journal of Algorithms*, vol. 7, no. 2, pp. 159–173, June 1986.

Symbolic computations in .NET Framework

Igor Medvedev, Yuri Okulovsky

Ural Federal University
Yekaterinburg, Lenina str. 51
Email: yuri.okulovsky@gmail.com

***Abstract*—The computer algebra system (CAS) is a software that is used for various symbolic computations like simplification and differentiation. CAS are based on the transformation rules that rearrange expressions according to the mathematical laws. We consider development of CAS in the .NET Framework. Currently, there is only one .NET software product with some features of symbolic computations, and no full-fledged implementation of the transformation rules in .NET. In the same time, the .NET framework provides many features for innovative techniques of rules' development, and therefore a .NET solution for the transformation rules can offer the new approaches to the computer algebra systems. In this paper, we describe these techniques and their implementation.**

***Index Terms*—symbolic computations, computer algebra systems, transformation rules, .NET framework**

I. Introduction

The computer algebra system is a software that performs symbolic computations. Typical examples of such computations are simplification of an expression into a smaller one, operations like differentitation and integration, logical interference and so on [1]. The computer algebra systems (CAS) are widely used in mathematics and computer sciences. In CAS, mathematical structures are represented in the symbolic form. It differs CAS from numerical analysis systems, which manipulate numbers. The most typical data representation in CAS is the syntax tree, an example of which is shown in Figure 1. Simplification, differentiation and other symbolic computations are performed as sequences of the elementary transformation rules, each rule rearranges a tree. The example of such transformations is given in Figure 2.

For the standard tasks, like simplification or calculus operations, many CAS systems are developed. In this area, the market has many solutions, including big enterprise packages like Mathematica, Maple or Mathlab, and small open-source projects. However, sometimes we need a computer algebra system not only to make a reasearch with its aid, but also to study CAS itself. For example, in [4] we propose a new genetic programming algorithm that combines simplification and induction as the uniformed parts of the evolutionary computations. To do so, we implemented the new transformation rules for mutation and crossover, and used them together with the simplification rules in the evolutionary algorithm. With the existing CAS, we would need the access to the system's core structures, because new rules should be programmed and merged with the existing ones, and then used in the completely new algorithm. For research of the transformation rules and computer algebra we need a different kind of CAS. The graphical user interface and the amount of the supported algorithms are not so important, while the easy and understandable access to the core structures is.

In this article, we present our approach to the transformational rules and the computer algebra algorithms. The most prominent distinction from the existing solutions is implementation in the C# language and the .NET framework [5]. The .NET Framework is a modern developing tool, widely used in science and education as well as in the commercial software development. .NET offers many convenient features, and some of them, like expression trees and lambda expressions, seem to be especially useful for the computer algebra. However, most of the existing CAS are programmed in old, maybe even

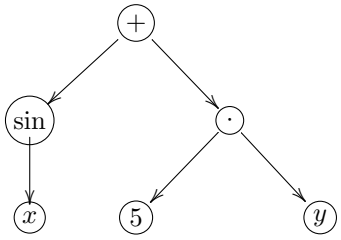


Fig. 1. Example of syntax tree for function $f(x, y) = \sin x + 5y$. Here x and y denote variables, 5 - a constant, and other nodes are operations of addition, multiplication and sine function.

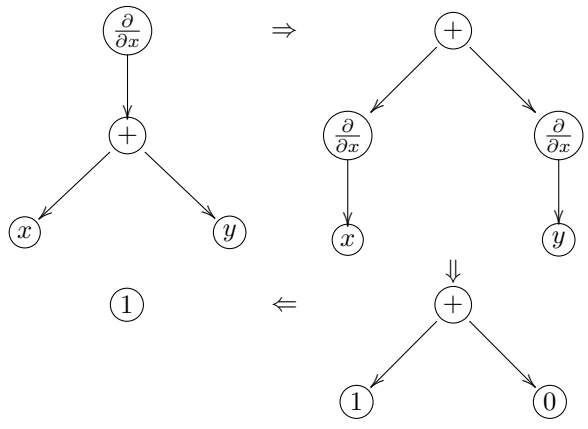


Fig. 2. A chain of transformations that computes $\frac{\partial(x+y)}{\partial x}$.

obsolete, languages like C, LISP or C++. We believe that the usage of the modern programming techniques for the computer algebra gives a fresh look on the symbolic computations, and could be resulted in the new approaches to computer algebra systems.

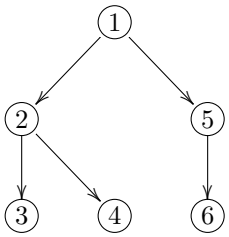
To our knowledge, only one .NET solution exists, named Math.NET [2]. However, it could hardly be considered as a full-fledged computer algebra system. The transformation rules are not programmed as a separate entity, and are substituted by Visitor pattern [3], that processes nodes in a tree according to its function. This decision hampers the system's expanding, because the addition of new operations demands alterations in the existing code. Moreover, even operations like differentiation of the exponential function are still not implemented. Of course, we do not need a new CAS system to perform simplification or differentiation when writing a .NET application. In most cases, we can just run the CAS application, perform all the necessary computations, and write the result back to the program. Or, we can use CAS system on the lower level and run the corresponding methods using .NET legacy code interoperation. However, seamless integration of computer algebra into .NET framework can be usable for some applications.

II. Rule definition in C#

Application of the rule can be subdivided into the following stages. In the *sampling* stage, the system that applies rules (which is called the *driver* below) selects some tree-like structure from the syntax tree, and presents its nodes as a tuple. In the *selection* stage, the driver sort out the tuples that do not meet the specified criteria. In the third stage, called *modification*, the driver transforms the tree according to the rule. In the most widespread case, the rule processes one tree. For such *unary* rules, the tree is rearranged with replacements. In some cases, the rule processes more than one tree. For example, the modus ponens rule in logical interference accepts two trees $A \to B$ and A and produces B. In this case, new tree is to be created from the selected nodes.

A. Sampling

To perform the sampling stage, we should specify the tree-like structure that we are searching for in a tree. Also, we need to map the nodes in the structure into a positions in a tuple. We used query strings of our own syntax to do that. Let us demonstrate the syntax of query string with the examples, shown in Figure 3.



A	(1)	The root of the tree
?A	(1), (2), (3), (4), (5), (6)	An arbitrary node in a tree
?A(B)	(5,6)	An arbitrary node with its unique child
?A(B,C)	(1,2,5), (2,3,4)	An arbitrary node in a tree with its two children in the fixed order
?A(.B,.C)	(1,2,5), (1,5,2), (2,3,4), (2,4,3)	An arbitrary node with its two children in the unconditioned order
?A(.B)	(1,2), (2,3), (2,4), (5,6)	An arbitrary node with its arbitrary child
?A(?B)	(1,2), (1,3), (1,4), (1,5), (1,6), (2,3), (2,4), (5,6)	An arbitrary node with its arbitrary descendant
?A(?B(?C))	(1,2,3), (1,2,4), (1,5,6)	An arbitrary node, its descendant and the descendant of the descendant
?A(?B(C,D))	(1,2,3,4)	An arbitrary node that has a descendant with two children

Fig. 3. Various examples of query strings. Queries are applied to the presented tree, its output is specified in the table.

The sampling algorithm is a depth-search algorithm that builds a correspondence between a given tree and a parse tree of a query string. Suppose the algorithm observes some node. It proceeds further according to the following rules:

- if the corresponding query substring has a form $(A_1, \ldots, A_n)$, the algorithm checks that the count of observed node's children is n, and assigns A_i to corresponding child.
- if the corresponding query substring has a form $(.A_1, \ldots, .A_n)$, the algorithm checks that the count of node's children is greater than or equals n. Then it runs through all possible assignments of A_i to observed nodes children. For all such assignments, further search will be performed.
- if the corresponding query substring has a form $(?A_1, \ldots, ?A_n)$, the algorithm assigns A_i to every possible combination of the node's descendants. For all such assignments, further search will be performed.

To encode the rule, we should specify the query string in the program. It could be done by encoding of the query as a constant string. However, it is not convenient due to the possible mistakes in the query's syntax, such as brackets mismatch or incorrect letters. To eliminate such errors, we implement query strings definition with square brackets overriding.

Consider the code in Listing 1 that specifies a query string. Here `Rules` is a static class that is purposed to create rules. Static method `Select` accepts a query and creates a `SelectClause` instance, which is used to define selection and modification, as is shown below. `AnyA`, `ChildB` and

Listing 1 Specification of sampling in the program.

```
public class  Creator : RulesCreatorBase {
  public void Create() {
    Rules.Select(AnyA[ChildB,ChildC])...
} }
```

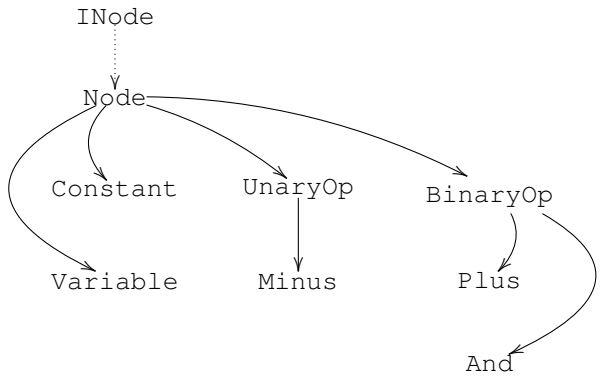


Fig. 4. A fragment of operators' type hierarcy, used to selection procedures.

`ChildC` are defined in `RulesCreatorBase` as properies that corresponds to $?A$, $.B$ and $.C$ elements of query strings.

B. Selection

The selection stage can be further subdivided into the type selection and the custom selection. The type selection checks the types of nodes in the selected tuple and rejects the tuple in the case of mismatch. The custom selection can check additional conditions, e.g. a value of a constant. The type selection must be performed prior to the custom selection, because the appliability of the custom selection depends on the node's type. For example, to check that the constant's value is zero, we have to be sure that the node corresponds to a constant, not to an operator.

The operations in our solution follows the type hierarchy that is shown in Figure 4. Each operation type has also its generic-analog that specifies the type of returning value. For example, `INode<T>` inherits `INode` and is implemented by `Plus<T>` that inherits `Plus`. Hence we can select operations either by their function (`Plus`), their returning type (`INode<T>`), or the combination of these properties (`Plus<T>`).

When programming selection, a challenge emerges of how to subject the tuple to the selection's condition. We cannot store selected nodes in the array or the list structures, because they do not allow specifying different types for elements. With the array representation, selection could only be performed in the following way:

```
array =>
    array[0] is Plus &&
    array[1] is Constant &&
    (array[1] as Constant).Value==0
```

Of course, constant casting and addressing is a potential cause of the type errors. We have developed an elegant solution for the selection with the aid of generic methods and code generation. Consider the code in Listing II-B. Here `SelectClause` is a class, which instances are created by `Rules.Select(...)` method. We can then call the `Where` method as shown inside the `Main` method. Note how naturally and easy-to-write this rule's definition is in comparison with casting of array elements.

Listing 2 Code pattern for selection stage, given on the example of three arguments

```
class WhereArg<T1,T2,T3> {
  public T1 A; public T2 B; public T3 C;}

class WhereClause<T1,T2,T3> {

Func<WhereArg<T1,T2,T3>,bool> selector;

public bool Where(object[] nodes) {
if (!(nodes[0] is T1)) return false;
if (!(nodes[1] is T2)) return false;
if (!(nodes[2] is T3)) return false;
var arg=new WhereArg<T1,T2,T3> {
   A=(T1)nodes[0],
   B=(T2)nodes[1],
   C=(T3)nodes[2] };
return selector(arg);
} }

public class SelectClause {
public static WhereClause<T1,T2,T3>
  Where<T1,T2,T3>
       (Func<SelectionArg2<T1,T2,T3>,bool)
  {...}
}

public class Creator : RulesCreatorBase {
public void Create() {
Rules.Select(A[B,C])
.Where<Plus,Constant,INode>
     (z=>z.B.Value==0);
} }
```

When we specify generic-arguments of `Where` method, we define the type selection that should be performed. If the type is not important, we just specify `INode`, since it is a basic interface for all nodes. Setting `Plus` as a type for the first element of the selected tuple allows us to specify a desired operation. Settings `Constant` as a type of the second element throws away all the tuples where the second element is not constant. Also, we may specify the custom selection condition `z.B.Value==0`, because the type of the second node is known to compiler after the type selection. Therefore, cast errors are catched on a compile stage. In addition, we can use the same letters for elements in selection that we have used

in sampling.

Declarations of `WhereArg` and `WhereClause` classes as in Listing II-B must be done for all different count of the arguments. We have used a code generation technique to produce declarations for up to 20 arguments, which is believed to be enough for our purposes.

Listing 3 Code pattern for modification stage, given on the example of three arguments.

```
class NodeDecorator<T> {
public T Node { get; private set; }
public void Replace(INode newNode) {  }
}

public class ModInput<T1,T2,T3> {
NodeDecorator<T1> A;
NodeDecorator<T2> B;
NodeDecorator<T3> C;
}

public class WhereClause<T1,T2,T3> {
public RuleInstance
   Mod(Action<ModInput<T1,T2,T3>> action)
      {...}
}

public class Creator : RulesCreatorBase {
public void Create() {
Rule.Select(A[B,C])
.Where<Plus,Constant,INode>
     (z.B.Value==0)
.Mod(z=>z.A.Replace(z.C));
} }
```

C. Modification

When the selection stage is over, we obtain several tuples that could be modified in the modification stage. However, only one of them will be actually processed, because application of the rule may invalidate other tuples. Therefore, the modification stage processes only one of the selected tuples. We have developed a "clean" modification, which does not affect the initial trees. Instead, in the modification stage we create a copy of the selected trees, and perform modification on the copy. To do that, we must find the roots of the nodes, presented in a given tuple, clone them, and further process a newly created tuple with a corresponding clones of the nodes.

For unary rules, modification turns into one or several replacements, which replace one of nodes with another. The method for replacement could not be placed in the `INode` interface, since it would give to the user an access to insecure replacements of the node. Therefore, we create a decorator class that wraps each node, and `ModInput` generic class that contains several decorators, as shown in Listing 3.

As we see, generated `WhereClause` class contains Mod method that processes a given typified tuple of decorators. Inside the given action, we have access both to the typified node (and therefore to the values and other type-specific content of the node) and to the Replace method. We can now declare a rule as in the Main method.

In case of not unary rules, we include the Produce method into WhereClause with the same approach: accept lambda that transforms ModInput into a node, and this node is considered as the root of the resulting tree.

III. Conclusion

The concepts above were successfully implemented and tested on various rules, mainly for simplification and differentiation. The computer algebra library for the .NET framework was written, with the following features:

- conversion of the .NET lambda expressions into the operation trees;
- simplification and differentiation of the .NET lambda expressions;
- linear regression of the .NET lambda expressions.

The developed rules were also successfully used in the genetic programming experiments, described in [4].

Acknowledgments.

The work is supported by the program of President of Russian Federation MK-844.2011.1.

References

[1] J. Grabmeier, E. Kaltofen and V. Wiespfenning. *Computer algebra handbook* Springer-Verlag, 2003

[2] Math.NET Project. http://www.mathdotnet.com/

[3] E. Gamma, R. Helm, R. Johnson and J. Vlissides *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley, 1994

[4] Ya. Borcheninov and Yu. Okulovsky *Genetic programming with embedded features of symbolic computations*, In Proceedings of international conference of Knowledge Discovery and Information Retrieval, 2011.

[5] A. Troelsen *Pro C# 2010 and the .NET 4 Platform* Apress, 2010

The Bufferbloat Problem and TCP: Fighting with Congestion and Latency.

Anatoliy Sivov
Yaroslavl State University
Yaroslavl, Russia
mm05@mail.ru

V.A. Sokolov (research supervisor)
Yaroslavl State University
Yaroslavl, Russia

Abstract—**The bufferbloat is a serious problem of the modern nets with complex organization. The persistent saturation of the net with an excessive buffering on hops results in a huge latency which can make interactive and realtime connections practically unusable. Most research in this area is devoted to Active Queue Management (AQM) which is used to keep the packet queue size correct on the bottleneck of the network path. This article gives the other look on the bufferbloat. It suggests to fight with the congestion – the reason of buffer saturation. It states that currently most widely used TCP loss-based congestion control algorithms have the problem which is shown in impossibility to correctly estimate the bandwidth in the presence of excessive buffering, and suggests to use the delay-based approach for congestion avoidance. It presents ARTCP which is delay-based and does not use burst sends limited by the congestion window, but transmits data on the adaptive rate estimated with pacing intervals.**

TCP; congestion; latency; bufferbloat; AQM; ARTCP.

I. Introduction

The bufferbloat is a term introduced by Jim Gettys which is very popular now and means the existence of excessively large and frequently full buffers inside the network. The bufferbloat influences the latency, an important parameter in the networking experience, very much. The latency consists of three kinds of delays: transmission delay, processing delay and queuing delay. The latter is the time the network packet spends waiting to be processed or transmitted. This time obviously depends on the queue size which can grow huge in the case of the bufferbloat.

The reason of this growth is a congestion the network path can suffer from. Paths between communicating endpoints are typically made up of many hops with links of different bandwidth. In the fast-to-slow transition hops we can have the situation when there are packets arrived to be queued or dropped, because they can not be immediately transmitted due to the previous arrivals being not processed (passed to the next hop) yet. The buffers (that carry the queue) are necessary in the "burst send" scenarios to maintain the flow of packets at the maximum rate and achieve the throughput. However, incorrect bandwidth estimation can lead to too many packets being sent, that results in buffers overpopulation or packets drops. If the buffers are too big, the packets are not dropped, and the queue grows, as well as the queuing delay. In this case we have unnecessary latency growth which does not lead to any benefits in throughput.

So, it is easy to conclude that in the presence of congestion in the net with excessive buffering the resulting delay can grow very high. As reported in [1], most network hops suffer from the excessive buffering. The other problem is that the network congestion is a usual case.

TCP is the most widely used transport protocol in the world. This protocol tries to use all the existing bandwidth to be efficient. To do that, it increments the transmission rate (actually, increases a congestion window) and provides the net with more and more data (that come out as burst sends limited by the congestion window) until a packet loss occurs. So, this protocol is one (and, possibly, main) of the reasons of congestion. TCP loss-based congestion avoidance algorithm surely uses the loss occurrence to slow down the transmission. This approach proved to be very efficient on reliable links with small buffers.

Today's networks generally do not meet the requirements listed above . With excessive buffering a packet loss in reliable links occurs on the complete saturation of the bottleneck buffer instead of the moment, when the congestion really occurred. This packet drop postponement results in bandwidth overestimation by TCP connection and consecutive packet losses that come from persistently full buffers. Moreover, this saturation of large buffers causes a delay increase, and big latency is surely experienced not on this TCP connection only, but on all network activities that share with it the same part of the network path.

To illustrate the situation, we can mention that 1 MB buffer is able to carry 8 seconds of data for 1 Mb/s connection. Provided that this is a bottleneck buffer and it stays full because of some big data transmission, we have 8 seconds queuing delay for any packet that passes this hop. This delay is much greater than common RTT for continental connections (about 100 ms). This huge latency makes interactive sessions (like ssh or web surfing) practically unusable and surely disallows the use of realtime networking like VoIP.

There are several approaches to fight with this problem. One of them is traffic prioritization or, more generally, Quality of Service (QoS) [2] which is out of scope of this article, because it does not help to improve the bufferbloat situation

The work is supported by the RFBR grant #11-07-00549-a

and does not decrease the latency for general applications. However, QoS is a very important mechanism which is used by realtime applications such as VoIP, IP-TV or online gaming.

The other approach is Active Queue Management (AQM) which attempts to set correct queue limits and either drop packets that overcome these limits or notify the sender about the congestion with ECN. These techniques try to eliminate the bufferbloat problem by removing the excessive buffering.

And finally, the last approach is to try to avoid the congestion without exhaustion of buffers. This is the most complex approach, because congestion can be of different nature. Despite the fact that congestion and excessive buffering are two different problems, and the best practice is to combine queue management and early congestion avoidance, congestion avoidance itself may solve the bufferbloat problem, if congestion is detected early, before buffers are saturated and latency heavily increased.

This paper consists of three sections. The first section very briefly discusses the current AQM solutions. The second section is devoted to the TCP congestion problem. It lists present-day TCP congestion avoidance algorithms and covers the latency increase problem. The last section introduces a version of ARTCP revised by the authors – the delay-based congestion avoidance algorithm that separates congestion avoidance and reliable delivery logic, and thus can be very efficient in lossy networks. The revised version does not use RTT for congestion avoidance, but uses pacing interval measurements only. So, it does not use indicators measured in the reverse direction and, unlike most other delay-based algorithms, behaves well, when a reverse direction path suffers from the congestion or delayed acknowledgments are used.

II. AQM SOLUTIONS

The problem of the excessive buffering is not new, it was discussed by John Nagle in 1985 [3] with an introduction of a "fairness" concept that suggested replacing a single FIFO with multiple queues (one per each source host) and service them in a round-robin fashion. This suggestion could limit the latency increase problem with one host and help to prevent a misbehaving host from making packet switch totally unusable for the others. Surely, there were no suggestions on single queue limitation and attempts to overcome the host bufferbloat problem.

The problem of bufferbloat became more serious today, when RAM is much cheaper and there are hops tuned for highspeed links (i.e. 1 Gb/s or even 10 Gb/s interfaces) that have to deal with relatively slow connections (about several Mb/s). So, this problem has been "rediscovered" today [1] and is actively discussed [4] by networking experts, giving a born for projects like bufferbloat.net.

Unfortunately, the excessive buffering can not be solved by simple static buffer management in several scenarios [4], and there is a need in Active Queue Management (AQM). The idea to detect congestion by the queue size and try to manage this size (and congestion) by dropping packets or marking them and, thereby, notifying the connection about congestion is not new. It is actively used in Random Early Detection (RED) algorithm presented by Van Jacobson and Sally Floyd in 1993 [5].

This AQM scheme monitors the average queue size and drops (or marks if ECN is used) packets based on statistical probabilities. It became a classic AQM implementation that was considered useful and strongly recommended for deployment in RFC 2309 [6]. Despite this fact, there is a strong criticism of RED [7, 8, 9, 12], mostly because of the difficulty in its configuration and the necessity of a rather large buffer in the bottleneck for RED to have time to react.

Nevertheless, RED is the most widely deployed and available AQM, which is the base for a wide variety of AQM algorithms. Some of them are presented in [10].

However, it should be mentioned that according to [4] "as much as 30 percent of residential broadband networks run without any queue management". Jim Gettys in his blog [11] names the main reason of "spotty deployment of AQM by network operators." It is a requirement of tuning and possibility of "trouble if not tuned properly". Also, he says with reference to Van Jacobson about two bugs in the classic RED discovered and proved by Kathy Nichols.

Van Jacobson considers [4], that RED can not be used with wireless nets because of the huge dynamic range presented by them. Also, he states there about the lack of the information helpful for determining the correct queue size RED can acquire: "I helped put people on the wrong track with RED which attempts to extract some information from the average queue size, but it turns out there's nothing that can be learned from the average queue size." He names the indicator used by BLUE and its successors "much more reliable" and pronounce his work with K. Nichols on RED Lite.

BLUE [12] is the AQM algorithm that uses the packet loss and link utilization history to manage congestion. BLUE maintains a single probability which it uses to mark (or drop) packets, when they are queued. If the queue is continually dropping packets due to a buffer overflow, BLUE increments the marking probability, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability.

BLUE, as any other single-queue AQM algorithm, treats all flows as a single aggregate. It brings unfairness to its behavior, because some single aggressive flow flooding the packets into the queue can push out some packets, that belong to other flows. To overcome this unfairness, the authors of BLUE presented the other AQM algorithm – Stochastic Fair Blue (SFB) which is a BLUE modification that hashes flows with a Bloom filter and maintains different mark/drop probabilities for every bin.

The other AQM algorithm, that attempts to save the protection provided by per-flow scheduling mechanisms, but combine it with its own simplicity, is RED-PD [13] which is the acronym for RED with Preferential Dropping. This is an AQM algorithm based on RED that uses the packet drop history at the router to detect high-bandwidth flows at the time of congestion and preferentially drops packets from these

flows. This AQM was designed to be effective at controlling high-bandwidth flows using a small amount of state and very simple fast-path operations in an environment dominated by end-to-end congestion control and a skewed bandwidth distribution among flows in which a small fraction of flows accounts for a large fraction of bandwidth.

Despite the intensive research in this area, there are still a lot of questions [4, 14, 15, 16] in AQM efficiency. Some of them are related to AQM performance, its behavior in heterogeneous, dynamic or unreliable environments; others consider security problems. One of such problems is DDoS attack vulnerability. Considerations of [17] states that the existing AQM algorithms are "rather vulnerable to spoofing DDoS attacks". However, this paper presents Resilient SFB (RSFB) algorithm as a solution of this problem, but there is surely the area for further research.

III. TCP CONGESTION AND QUEUING

TCP is the most widely used transport protocol. Designed to efficiently consume available bandwidth of dynamic nets with an unknown topology, it tries to achieve the goal by continuous monitoring of the network capacity. Until a packet loss occurs, TCP constantly increases its congestion window as a reaction on the acknowledgment. Unless limited by a receiver's window or application providing data for transmission, it will supply a link with more and more data being sent in a burst limited by the congestion window.

This technique is used to determine the bandwidth, starting with a small value and increasing it until a drop caused by congestion happens. As a reaction on the drop, TCP decreases its congestion window (halves it or drops to initial value depending on the use of "fast recovery"). After that, it increases its window again. The idea of this balancing determined by four intertwined essential TCP algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery – defined in [18], is to keep the network operating near the inflection point, where the throughput is maximized, the delay is minimized, and a little loss occurs.

This point is known to be a bandwidth-delay product (BDP), the value that corresponds to the amount of data "in flight" exact for total link capacity consumption without congestion. BDP was a recommendation for the buffer size, and internet researchers and engineers had to advocate for sufficiently large buffers [1], because TCP uses burst transmissions and the lesser buffers could result in heavy losses.

However, these buffers lead to another problem: with their presence, they became a part of a "pipe" that TCP tries to fill to be efficient. The drop indicator used to estimate the available bandwidth triggers, when the bottleneck buffer is full. It results in three problems: TCP overestimates the link throughput, the latency increases due to the growth of the queuing delay, there is no space in full buffer to absorb the burstiness of the network.

Despite this fact, firstly recognized in 1989 [1], the packet loss remains to be the congestion indicator in modern TCP implementations. So, CUBIC [19] is a default congestion control algorithm in Linux. This congestion avoidance algorithm is loss-based, and thus it suffers from the pointed problems. For instance, [1] demonstrates the bufferbloat problem in CUBIC.

The reasons of such a behavior are understandable, and there is a strong interest in better behaving algorithms that take the delay indicator into account. The problems of existing such an efficient congestion control were, for example, discussed at the Linux Plumber Conference in 2011 [21], where the suggestion about refining a delay-based portion of hybrid congestion control algorithms was declared.

The idea of a delay-based congestion control is apparently not new, it was considered by Raj Jain in 1989 [22]. He suggested using a black-box model of the network, where changes in round-trip delays can be used as an implicit feedback, and stated that achieving congestion avoidance in heterogeneous networks is difficult, because the congestion feedback from one subnetwork may have no meaning to sources on other subnetworks.

The delay-based congestion control is promising for the networks where the latency is important, because delay-based algorithms tend to minimize queuing delays. It includes the networks with an active use of realtime data connections like VoIP or interactive sessions where latency degrades experience very much. Also, the connections with delay-based congestion control are known to be more efficient than loss-based in scenarios, where many connections share the same network path, because of being more fair and thus being more capable in an effective cooperation consuming a limited resource. However, these algorithms are less aggressive (that is considered as their plus from one point of view) and so, sharing the link with the greedy loss-based algorithm (like TCP CUBIC), they will get the less part of an available bandwidth.

These considerations underlie the growing interest in hybrid congestion avoidance algorithms. The research in this area resulted in the appearance of several hybrid congestion control algorithms in the second half of 2000's. Microsoft Research introduced Compound TCP [20] in 2005. And University of Illinois developed TCP-Illinois [23] in 2006. Both algorithms were designed for high-speed links as a reaction of TCP Reno too slow bandwidth utilization. They are rather aggressive and neither TCP-Illinois (see 'Background and motivation' section of [23]) nor Compound TCP (see Section III of [20]) were designed with a small latency goal in mind. Moreover, they are considered [24] rather unfair in some circumstances.

The other interesting congestion avoidance algorithm, that is to avoid latency increase problem, is TCP-NV [25] presented by a host networking team member from Google, the main developer of TCP Vegas [26] Lawrence Brakmo in 2010. The goal of this algorithm is to prevent queue build-up and packet loss, while making full utilization of the available bandwidth. TCP-NV is based on TCP-BIC [27] and behaves like it, unless the queue build-up is detected. On queue build-up detection, it decreases the congestion window proportionally to the amount of the detected build-up. The main problem in this approach, named by Brakmo, is a noise in RTT measurement caused by

such optimization techniques as TSO, LRO, and interrupt coalescence. Since TCP-NV is optimized for data-centers, it has to deal with these hardware tricks inherent to 10G NICs and has to filter affected measurements with various quirks.

The other problem RTT-based congestion control engines suffer from is the reverse congestion which results in ACK delivery delays and thus can be falsely treated by these engines as forward congestion indication. This problem was mentioned by Brakmo at the Linux Plumbers Conference in 2010 and is well illustrated for TCP Compound in [24]. Brakmo's suggestions for this problem were to prioritize pure ACKs in host and switches/routers and to measure reverse delay with the help of a new TCP option and adjust appropriately.

Besides hybrid congestion avoidance algorithms, there are some delay-based that worth mentioning. The most famous delay-based algorithm is TCP Vegas [26]. It can not rival with loss-based algorithms on the same link due to its lesser aggressiveness, does not fit high-speed links, is not efficient in nets with very variable delays and can lead to persistent congestion in some circumstances [28]. However, it is a sign algorithm, that showed vitality of the delay-based approach and gave a birth to the whole family of delay-based congestion avoidance algorithms.

FAST TCP [29] is one of Vegas-derived algorithms. It is a delay-based algorithm designed for networks with large bandwidth-delay product with responsiveness in mind, that seeks to maintain a constant number of packets in queues throughout the network by measuring the difference between the observed RTT and the “base RTT”, which is RTT observed without queuing. It is a promising algorithm, but its fairness is very sensitive to correct “base RTT” estimation [30] and, being TCP Vegas descendant, it suffers from some inherent problems [28]. Another issue of FAST TCP, because of which it can not be widely used, is patent restrictions imposed by its authors.

The other attitude to delay-based congestion avoidance is made by the authors of a new CDG algorithm [31] developed by CAIA, Swinburne University of Technology (Melbourne). They claim that the biggest limitation of the most delay-based congestion control algorithms is to infer congestion, when the path delay hits or exceeds certain thresholds. This approach suffers from the circumstance, that it is very hard to set the meaningful threshold, unless there is a reliable information about the path that packets will take. (It is mentioned above, for an instance, that FAST TCP suffers from relative unfairness in case of inability to accurately estimate path “base RTT” or TCP Vegas causes persistent congestion, when its threshold is set incorrectly.) They suggest that the threshold should be abandoned and “delay-gradient” congestion control, that relies on relative movement of RTT and adapts to particular conditions along the paths each flow takes, be used. The CDG algorithm is very interesting and seems to utilize the available bandwidth rather efficiently compared with loss-based CUBIC and NewReno algorithms [32]. However, available results of tests are ambiguous (especially from latency point of view) and there are few experiments to make a conclusion.

One of the queuing reasons (excluding congestion, when any queue will obviously become full) is the burstiness. To decrease this effect, TCP pacing may be worth applying. TCP pacing is the technique consisting in spreading the transmission of TCP segments across the entire duration of the estimated round trip time instead of having a burst at the reception of acknowledgments from the TCP receiver. Despite offering better fairness, throughput and low drop rate in some situations, pacing may be dangerous and, contrary to intuition, can cause significantly worse throughput because of congestion signal postponement and synchronized losses [33]. However, these results vary for different congestion control algorithms and pacing rates. From [34] it can be concluded that pacing can cause throughput degradation in mixed (both paced and non-paced flows) scenarios for various loss-based congestion control algorithms. On the contrary, [34] shows the better characteristics for delay-based congestion control algorithms, when pacing is used, including fairness enhancements due to the effect of reducing queue variation.

Let us summarize the above observations on TCP congestion and latency. Loss-based congestion avoidance algorithms saturate the bottleneck queue, that results in a great deal of queuing delays in the case of excessive buffering. Hybrid algorithms are promising in competing with loss-based in bandwidth utilization. However, their fairness must be revised, and they have a danger of issuing problems of both loss-based and delay-based approaches in different scenarios depending on the way, in which they combine loss-based and delay-based portions. In general, delay-based congestion control algorithms obviously have the best latency. These algorithms issue several difficulties: if they are designed to rely on a certain threshold, this threshold must be correctly estimated, otherwise the errors in estimation may result in algorithm work degradation and even persistent congestion, delay-based algorithms need rather precise time interval (i.e. RTT) measurement, the work of delay-based algorithm that uses RTT may be severely degraded by the reverse congestion, so forward-only indicators like “single trip times” are much more reliable. TCP pacing may lead to a worse behavior for loss-based congestion avoidance algorithms and even increase latency, but it improves characteristics of flows controlled by delay-based algorithms.

IV. Introducing ARTCP

Taking into account the above considerations, we introduce a revised variant of ARTCP – the TCP modification with delay-based congestion control that uses pacing and pacing intervals, which is a low latency loss-insensitive solution with a good throughput.

Adaptive Rate TCP (ARTCP) [35] does not use the congestion window at all. Instead, it uses pacing intervals measurement, so called “duty factor”, to determine the congestion and control the data flow speed. The idea of ARTCP is to remove burstiness by applying pacing with an adaptive rate. The pacing here is a key element, because it determines a sender's speed, and the analysis of its alteration is used to detect the congestion. To achieve this goal, we calculate pacing intervals in two points. The first point is a sender itself, here, pacing interval is just a time difference between the transmission of two successive segments, which is

determined by currently used transmission rate. The second point is a receiver. On its side, we calculate the time difference between two successive arrivals and send it back to the sender with an acknowledgment [36]. To prevent incorrect calculation of pacing intervals the sender provides the receiver with additional information, so called a “packet sequence number” (PS), so that the receiver could determine (invisible for it in some cases without this information) losses and packet reordering efficiently, and thus filter out packets that are not successive and avoid incorrect estimation of pacing intervals.

We treat these pacing intervals estimation as measures of transmission and reception rates. Thereby, we can use them on the sender as a congestion indicator. The situation when the reception rate is stabilized in the presence of the growing transmission rate can be certainly taken in as congestion. This variant is attractive comparing with the fallback RTT-based delay estimation, because it strictly uses forward direction time measures only, and thus, it is totally insensitive to the reverse congestion which causes troubles for most delay-based congestion avoidance algorithms.

Because ARTCP is not so sensitive to reverse direction delays, it has no limitations of RTT-based congestion avoidance algorithms that need immediate acknowledgment dispatch and behave badly, if delayed acknowledgments are used. So, with ARTCP we can delay acknowledgment in case receiver is expecting, that it will transmit data soon, or it is waiting for a start of the scheduled data transmission, and piggyback this acknowledgment, as well as pacing interval information, with these data, thus increase the efficiency of bandwidth utilization.

To behave well, ARTCP must follow the several states: slow start, initial multiplicative decrease, recovery, fine tuning, and multiplicative decrease. Slow start stage is to rapidly determine the available bandwidth. This state is the first after the connection setup, and there the sender increases its transmission rate exponentially. Here, we have to deal with two contradictory challenges: the first one is to occupy the available bandwidth as soon as possible, the second is to prevent big latency growth which is inescapable, if the slow start caused a congestion. Congestion is mandatory in order to find the throughput limit. Our goal is to examine it as early as possible and to drop the transmission rate accordingly to compensate the congestion.

Compared with loss-based TCP congestion avoidance algorithms, ARTCP has a better chance to achieve the smaller transmission rate oscillation in the slow start, because it has no need to saturate the bottleneck queue to discover the congestion. Instead, it examines the pacing intervals sent back by the receiver and checks whether they are not increasing, while transmission rate still continues increasing. Since we found the transmission rate we can rely, estimated with pacing intervals taken from the receiver, we have a bandwidth estimation. The current transmission rate will be surely greater than the estimated rate, because congestion is mandatory. So, we must compensate the congestion by reducing the transmission rate to the value that is less than the estimated one.

To perform the compensation, we do the initial multiplicative decrease which reduces the transmission rate, and goes to the recovery state in order to return to the estimated rate, when congestion compensation is over. Congestion compensation here is the amount of “superfluous” packets we sent, while the transmission rate is greater than our estimation. So, we must set the transmission rate lower than the estimation and increase it to the estimation in the way that allows the exact compensation we need.

It is a difficult question to what value it's better to drop the rate and how then it should increase. As a current solution, we suggest setting the rate as low as necessary, so that the recovery finishes as soon as possible, and increasing it linearly. This will allow the bottleneck queue size to decrease more rapidly, and thus have a propitious impact on the experienced latency. While linear increase is performing, we must constantly monitor the receiver's feedback, because the estimated bandwidth can be reduced by the other connection activities. The lack of doing that will result in bandwidth overestimation and in this case it will bring the network to the congestion. To detect bandwidth reduction in the recovery state, we can examine, whether the receiver's pacing intervals do not go down, while we proceed with transmission rate increasing. Since we achieved the estimated rate or the new estimation is discovered, we accept this rate as transmission rate and move to a fine tuning state.

Fine tuning is the most challenging state. It must adjust the transmission rate to changes in the available bandwidth and be able to rapidly react to heavy changes (decreases and increases). We suggest making adjustments in a probabilistic manner, and at the same time have the guards that can detect big changes of available bandwidth. Unless delay increase is detected (increase of the receiver's pacing intervals is observed), we set positive transmission rate speedup probability, so that the transmission rate can be increased. This allows us to increase the transmission rate, when the available bandwidth grows.

There is an interesting question, how we can inspect that the available bandwidth heavily increased. As a solution, we can increase the speedup portion, while we have a positive feedback from these actions, and reset it on any negative feedback. Applying these actions, we must be very careful not to provoke a congestion. To deal with this problem, we may address to the speedup portion increase only after we are assured about absence of negative impact on latency with acknowledgments of the data sent after previous increase. However, this speedup portion manipulation needs a further investigation, and we currently do not implement it, but deal with the probability of (mainly) a linear speedup.

At the same time we are looking for a delay increase. If it is detected, we set up slowdown probability, instead of speedup. As opposed to speedup, slowdown portion can be proportional to the current transmission rate, that makes us rather rapidly react on latency increases. In any case we must provide two mechanisms for slowdown: one – for a small tuning of the transmission rate, and another – for multiplicative decrease. The first is applied when we have a small receiver's pacing interval increase or have no decrease after the transmission rate

was increased. We call it “speedup undoing”. The other mechanism must be invoked if we observe big latency increase which is considered as a heavy congestion. In this case ARTCP yields the portion of its bandwidth share by exponential transmission rate drop. This drop is done by multiplicative decrease state which divides the current transmission rate by a certain ratio and then checks, whether it solved the latency problem. If not, it continues multiplicative decrease, otherwise it passes the connection to the fine tuning state.

This ends up a concise overview of ARTCP flow control states. As it is presented above, ARTCP uses the only indicator of congestion: latency increase which is examined by calculation of forward direction characteristics only. So, ARTCP is insensitive to packet losses and reverse congestion, what makes its idea very attractive. It is designed for lossy networks and solutions that require low latency, and it is supposed to be the right approach to the solution of the bufferbloat problem for links, where congestion is caused by TCP traffic. The main limitation of ARTCP is that it is a sender+receiver solution and it requires that both ends implement ARTCP, in the contrary to many sender-only TCP congestion avoidance solutions. However, ARTCP is designed to make it possible to use any TCP congestion avoidance algorithm as fallback and to switch to it on the fly, if the other end does not implement ARTCP [36].

[1] J. Gettys, K. Nichols, “Bufferbloat: Dark Buffers in the Internet” // Magazine Queue – Virtualization, vol. 9, issue 11, New York: ACM, 2011

[2] M. Marchese, QoS over heterogeneous networks, N.J. : John Wiley & Sons, 2007

[3] J. Nagle, On Packet Switches With Infinite Storage. // RFC 970, 1985.

[4] V. Cerf, V. Jacobson, N. Weaver, J. Gettys, “BufferBloat: What's Wrong with the Internet?” // Magazine Queue – Log Analysis, vol. 9, issue12, New York: ACM, 2011

[5] S. Floyd, V. Jacobson, “Random Early Detection gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413

[6] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshal, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, Recommendations on Queue Management and Congestion Avoidance in the Internet. // RFC 2309, 1998.

[7] V. Jacobson, K. Nichols, K. Poduri, “RED in a Different Light”, unpublished, 1999.

[8] M. May, J. Bolot, B. Lyles, “Reasons not to deploy RED”, technical report, 1999

[9] S. Harhalakis, N. Samaras, V. Vitsas, “An Experimental Study of the Efficiency of Explicit Congestion Notification” // PCI '11 Proceedings of the 15th Panhellenic Conference on Informatics, pp. 122-126

[10] S. Floyd, References on RED (Random Early Detection) Queue Management, web page, 2008 // url: http://www.icir.org/floyd/red.html

[11] J. Gettys, blog of the author of bufferbloat.net project, web page // url: http://gettys.wordpress.com/

[12] W. Feng, D. Kandlur, D. Saha, K. Shin, “BLUE: A New Class of Active Queue Management Algorithms”, U. Michigan Computer Science Technical Report (CSE–TR–387–99), 1999

[13] R. Mahajan, S. Floyd, D. Wetherall, “Controlling High-Bandwidth Flows at the Congested Router”, Proceedings to 9th International Conference on Network Protocols (ICNP), 2001

[14] S. Bohacek, K. Shah, G. Arce, M. Davis, “Signal Processing Challenges in Active Queue Management” // IEEE Signal Processing Magazine, 2004, pp. 69-79

[15] K. Shah, S. Bohacek, E. Jonckheere, “On the performance limitation of Active Queue Management (AQM)” // 43rd IEEE Conference on Decision and Control, 2004, pp. 1016-1022

[16] A. Kuzmanovic, “The power of explicit congestion notification” // ACM SIGCOMM Computer Communication Review, vol. 35, issue 4, New York: ACM, 2005

[17] C. Zhang, J. Yin, Z. Cai, “RSFB: a Resilient Stochastic Fair Blue algorithm against spoofing DDoS attacks” // International Symposium on Communication and Information Technology (ISCIT), 2009

[18] M. Allman, V. Paxson, E. Blanton, TCP Congestion Control // RFC 5681, 2009

[19] S. Ha, I. Rhee, L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant” // ACM SIGOPS Operating Systems Review, vol. 42, issue 5, 2008

[20] K. Tan, J. Song, Q. Zhang, M. Sridharan, “Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks” // 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006

[21] S. Hemminger, “A Baker's Dozen of TCP bakeoff?” // Linux Plumbers Conference, 2011

[22] R. Jain, “A Delay Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks” // Computer Communications Review, ACM SIGCOMM, 1989, pp. 56-71

[23] S. Liu, T. Basar, R. Srikant, “TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks” // Proceedings of the 1st international conference on Performance evaluation methodolgies and tools, 2006

[24] D. Leith, L. Andrew, T. Quetchenbach, R. Shorten, K. Lavi, “Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms” // Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2008), 2008

[25] L. Brakmo, “TCP-NV: Congestion Avoidance for Data Centers” // Linux Plumbers Conference, 2010

[26] L. Brakmo, L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet” // IEEE Journal on selected Areas in communications, vol. 13, 1995, pp. 1465-1480

[27] L. Xu, K. Harfoush, I. Rhee, “Binary Increase Congestion Control for Fast, Long Distance Networks” // INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2514-2524

[28] R. La, J. Walrand, V. Anantharam, “Issues in TCP Vegas”, UCB/ERL Memorandum, No. M99/3, UC Berkeley, 1999

[29] C. Jin, D. Wei, S. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance” // INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2490-2501

[30] L. Tan, C. Yuan, M. Zukerman, “FAST TCP: Fairness and Queuing Issues”, IEEE Communications Letters, vol. 9, no. 8, 2005, pp. 762-764

[31] D. Hayes, G. Armitage, “Revisiting TCP Congestion Control using Delay Gradients” // Proceedings of 10th International IFIP TC 6 Networking Conference, 2011

[32] G. Armitage, “A rough comparison of NewReno, CUBIC, Vegas and ‘CAIA Delay Gradient’ TCP (v0.1)”, CAIA Technical report 110729A, 2011

[33] A. Aggarwal, S. Savage, T. Anderson, “Understanding the Performance of TCP Pacing” // Proceedings of the IEEE INFOCOM 2000 Conference on Computer Communications, 2000, pp. 1157 – 1165.

[34] D. Wei, P. Cao, S. Low, “TCP Pacing Revisited” // Proceedings of the IEEE INFOCOM 2006 Conference on Computer Communications, 2006, pp. 56-63

[35] I. V. Alekseev, V. A. Sokolov Compensation Mechanism for Adaptive Rate TCP. // 1-St International IEEE/Popov Seminar "Internet: Technologies A and Services". P. 68-75, October 1999

[36] A. Sivov, “ARTCP packet structure. Features of the Formation and Processing of ARTCP Headers in Linux Network Subsystem” // Modeling and Analysis of Information Systems, vol. 18, №2, Yaroslavl: Yaroslavl State University, 2011, pp. 129-138

Detecting faults in TFTP implementations using Finite State Machines with timeouts

M. Zhigulin, S. Prokopenko, M. Forostyanova
Radio-physical department
Tomsk State University
Tomsk, Russia
maxzh81@gmail.com, s.prokopenko@sibmail.com,
mariafors@mail.ru

***Abstract*—In this paper, we consider a test derivation strategy for testing protocol implementations based on Finite State Machines with timeouts. The strategy is applied for testing TFTP implementations.**

Keywords-Finete State Machine (FSM); FSM with timeouts (timed FSM); transition tour

I. INTRODUCTION

FSM-based test derivation strategies for conformance testing of protocol implementations are well known [see, for example, 1-3] and a number of formal methods were developed for deriving tests which check time constraints of a discrete event system implementation. Some of them use FSM-based strategies for test derivation [4-11]. One of such strategies uses the model of a timed FSM (TFSM) with so-called timeouts [7-11], i.e., if no input is applied during an appropriate time period the FSM can move to another prescribed state. Correspondingly, the behavior of an FMS significantly depends on a time instance when an input is applied, i.e., the behavior of the FSM is specified for timed input sequences. In [12], it is shown how this behavior can be described by an ordinary FSM with an additional input symbol (a time unit) and thus, despite of the fact that a test suite derived for such an abstract FSM using black-box testing methods returns highly redundant tests, FSM-based test derivation methods can be directly used when deriving tests from an FSM with timeouts. In this paper, we derive a test suite as a transition tour of an appropriate FSM, since W-based testing methods [3] ask for the specification FSM to be complete and deterministic and this usually does not hold for FSMs which describe protocol behavior. We then analyze the fault coverage of a transition tour for TFTP (Trivial File Transfer Protocol) [13] implementations where the behavior significantly depends on timeouts. The contributions of the paper can be summarized as below.

- In this paper, we manually extract an FSM with timeouts from the description of TCTP, transform the obtained TFSM into a classical FSM and derive a test suite as a transition tour of the latter.
- The obtained test suite is then applied to available TFTP implementations and the set of detected faults is described.

The structure of the paper is as follows. Section 2 contains the preliminaries for FSMs with timeouts (TFSMs) and a brief description of the transformation of such TFSM into a classical FSM. In Section 3, the TFCP behavior as a timed FSM is presented and the set of faults which were detected in protocol implementations using a derived test suite is described. Finally, we conclude the paper in Section 4.

II. FINITE STATE MACHINES WITH TIMEOUTS

In this paper, we consider Finite State Machines (FSMs) which are augmented with timeouts in order to explicitly take into account timed aspects of the system behavior. A *timed FSM* S (or TFSM) is a 6-tuple $(S, I, O, \lambda_S, s_0, \Delta_S)$, where S, I and O are finite non-empty sets of states, inputs and outputs, respectively, s_0 is the initial state, $\lambda_S \subseteq S \times I \times O \times S$ is a transition relation, Δ_S: $S \to S \times (N \cup \{\infty\})$ is a *timeout function* where N is the set of nonnegative integers.[1] The function Δ_S has two projections $\Delta_S(s)_{\downarrow N}$ and $\Delta_S(s)_{\downarrow S}$. If no input is applied at a current state s before a timeout $\Delta_S(s)_{\downarrow N}$ expires then the TFSM will move to another state $\Delta_S(s)_{\downarrow S}$ as it is prescribed by the timeout function. If $\Delta_S(s)_{\downarrow N} = \infty$ then the TFSM can stay at the state s infinitely long waiting for an input, i.e., in this case, $\Delta_S(s) = (s, \infty)$. As usual, the notation $s - i/o \to s'$ is used for a 4-tuple $(s, i, o, s') \in \lambda_S$ while using $s - t \to s'$ for a triple $\Delta_S(s) = (s', t)$.

Example. Consider a TFSM in Figure 1. If an input *ACK*_3 is applied at state Wait2 at time instances 0, 1, 2 then the TFSM produces the output *ERROR* and moves to state Init. However, if an input *ACK*_3 is applied at state Wait2 at time instance 3 then the TFSM is at state Init and thus, input *ACK*_3 cannot be applied, since a transition under this input is not specified at state Init.

If for each pair $(s, i) \in S \times I$, there is at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in \lambda_S$ then the TFSM is *deterministic*; otherwise, the TFSM is *nondeterministic*. If for each pair $(s, i) \in S \times I$, there is at least one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in \lambda_S$ then TFSM is *complete*; otherwise,

[1] An abstract output can contain a positive integer for describing a delay of producing an output after an input is applied.

the TFSM is *partial*. The FSM in Figure 1 is partial and nondeterministic.

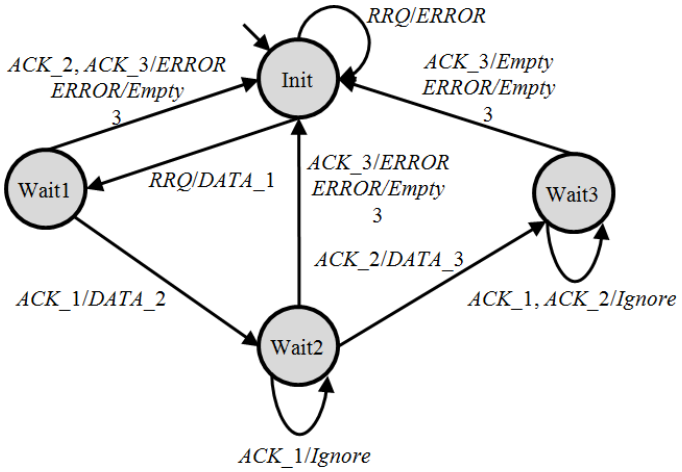


Figure 1. A TFSM for TFTP.

In order to describe the TFSM behavior we use the notion of a timed input that indicates that an input i is applied at time t. Correspondingly a *timed input* is a pair $(i, t) \in I \times Z_+$ where Z_+ is the set of nonnegative integers. In order to extend the transition relation to timed inputs we have to know at which state s' is an FSM when applying an input i at time t and the state s' is determined based on the timeout function [12]. Correspondingly, there is a transition $(s, (i, t), o, s'')$ if and only if there is a transition $(s', i, o, s'') \in \lambda_S$. The behavior of the TFSM is then extended to timed input sequences in a usual way. Consider the TFSM in Figure 1, state Wait2, input *PRQ* and $t = 5$. At time instance 3 the FSM will move to state Init with the timeout ∞ and thus, there is a transition (Wait1, (*PRQ*, 5), *ERROR*, Init) in the TFSM. Given a timed input sequence α, a pair (α, β), $\beta \in O^*$, is a *timed trace* at state s if there exists state s' such that $(s, \alpha, \beta, s') \in \lambda_S$. The set $out_S(s, \alpha)$ is the set of all possible output responses of the TFSM at state s to the timed input sequence α: $\beta \in out_S(s, \alpha)$ iff α/β is a timed trace at state s.

Given a TFSM S, state s of the TFSM is an *input-reachable* state (*ir*-state) if there exists a timed trace (α, β) such that $(s_0, \alpha, \beta, s) \in \lambda_S$ [12]. If a state is not input reachable then it is an *input-unreachable* (*iur*-state). *Iur*-states are not stable: we only can implicitly check that a TFSM has passed this state. The TFSM is *connected* if each state s' is an *ir*-state or there exists an *ir*-state s such that $s' = time_S(s, t)$ for some t. In this paper we consider only connected FSMs. Given a TFSM S, a finite set *TS* of timed input sequences is a *transition tour* of S if for each connected submachine B of S, each *ir*-state b of B and each input i specified at state b, it holds that the set *TS* has a timed input sequence $\alpha.(i, 0)$ such that α takes the TFSM B from the initial state to state b.

Given a transition tour *TS* of the TFSM *S*, the test suite *TS* can detect all output faults at each *ir*-reachable state. Moreover, as we illustrate in the next section, such a test suite can also detect other functional faults. A transition tour of a TFSM can be derived in different ways. A TFSM can be transformed into a classical FSM in such a way that there is the one-to-one correspondence between timed traces of the TFSM and traces of the abstract FSM with the tail input symbol different from 1 [12]. In this case, similar to [4], a special input 1 that corresponds for waiting 1 time unit is added. Since an FSM has to provide an output to each input, we also add a proper output N that corresponds to the case when the FSM does not produce anything. If a timeout at state is more than 1 then we add copies of the next state: the number of copies equals to the value of timeout minus 1. For each other input, there is the same transition at a copy of a state s as at the original state s. If there is an output delay T for a transition i/o then we consider a timed output (o, T) instead of the output o.

In this paper, given a TFSM we derive a test suite *TS* using the following steps.

Step 1. Derive a corresponding classical FSM adding the designated input 1 (*WAIT_1_time_ instance*) and output N.

Step 2. If a derived FSM is nondeterministic derive a set of deterministic submachines in order to cover each transition of the initial FSM.

Step 3. For each deterministic submachine:
- derive a test suite as a transition tour;
- replace each chain $1\ldots1i$ of k transitions $1\ldots1$ and an input i as a timed input (i, k);
- merge all the test suites into a single test suite *TS*.

For each conforming TFTP implementation P and each input sequence $\alpha \in TS$, it holds that $out_P(p_0, \alpha) \subseteq out_S(s_0, \alpha)$. If for some input sequence $\alpha \in TS$, it holds that $out_P(p_0, \alpha) \not\subset out_S(s_0, \alpha)$ then an implementation under test is faulty and the fault is detected by the test suite *TS*.

III. Testing TFTP implementations

The TFTP brief description is taken from [13]. The TFTP (Trivial File Transfer Protocol) is a simple file transfer protocol and it can read and write files (or mail) from/to a remote server. At the first step, there is a request for connection and when the connection is opened the file containing a sequence of blocks of 512 bytes can be sent. If a data packet has less than 512 bytes then there are no packets to be transferred. Each packet contains one data block, and must be acknowledged before the next packet can be sent. A loss of a data packet is prevented by the timeout functions; a lost packet can be retransmitted and the sender has to keep just one packet at hand for retransmission. According to RFC, both connected machines are considered as senders and receivers: one sends data and receives acknowledgments; the other sends acknowledgments and receives data. When an error occurs an error packet is sent. Most errors cause the termination of the connection, and timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events. The request cannot be satisfied, i.e., the file is not found, there is access violation etc. Another error type occurs when an incorrectly formed packet is received or there is no access to necessary resources. The only error condition that does not cause termination is a situation when the source port of a received packet is incorrect. This protocol is very restrictive, in

order to simplify implementation. Nevertheless, its available implementations still do not conform to the specification.

Five kinds of packets are supported:

- Read request (RRQ);
- Write request (WRQ);
- Data (DATA);
- Acknowledgement (ACK);
- Error (ERROR).

Packages RRQ and WRQ are sent by a client for connection establishment. The connection is established if after sending a packet RRQ (WRQ) the server replies with the first data packet DATA_1 (or after the acknowledgement ACK_0 to zero data packet). Each data packet DATA has the designated number that is an integer started from 1. Correspondingly, when acknowledging the receipt of a data packet a recipient sends the response ACK with the same number. Thus, when the server sends the response ACK_0 as a reply to the request to read the file this only means that the request is confirmed and the server is ready to transfer data. If the server replies with ERROR then the connection is broken and a client has to ask for the connection again. The server uses a special port for requests that is 69 by default. When getting a positive response the server a new port is randomly assigned for data transmission. The latter allows other clients using the main port.

In order to simplify the TFSM that is obtained when describing the protocol behavior we will test only the part that is responsible for getting files from the server (the request RRQ) under the following assumptions. Not more than three packages are transferred as a sequence, no retranslation is allowed and the timeout for waiting a packet equals three seconds. As we show below, even such limitations allow the derivation of a TFSM based test suite that detects faults in known protocol implementations.

Figure 1 represents a TFSM that describes the TFTP behavior when getting the request RRQ. State Init is the initial state. At this state only one input is specified that corresponds to the request to read the file. According to the protocol description, when the first part of DATA (DATA_1) is sent after the request and the FSM enters state Wait1. If there is no such a file or no access to the file, then the server responses with the message about an error (ERROR) and returns to the state Init. The behavior is mostly the same at states Wait1, Wait2, Wait3: the system is waiting for the acknowledgement of the sent data packet. If there is no acknowledgement after 3 seconds then the connection is broken (since due to the above restrictions, the retranslation is not allowed in the simplified TFSM description). Otherwise, the second packet DATA_2 is sent and the system enters state Wait2. If this packet is acknowledged (there is the input ACK_2) then the third packet Data_3 is sent and the system enters state Wait3. If there is the acknowledgement ACK_3 then all the packets have been received and the connection can be closed. Duplicated acknowledgements are ignored by the server.

A transition tour derived for the FSM in Figure 1 has total length of 674 inputs. This has been used for testing some TFTP implementations.

The following implementations being tested:

- class **TFTPServer** defined in the commons-net-2.0.0 library developed by Apache;
- **atftpd** Linux server developed by Jean-Pierre Lefebvre.

Both implementations have been declared to support TFTP [13]. However, the derived FSM with timeouts only partially describes the behavior of the **adftp** server, since there is no parameter in the implementation for changing the number of retranslations. Some mismatching has been detected between the protocol specification [13] and an implementation under test.

When testing the class **TFTPServer** defined in the commons-net-2.0.0 library the following mismatching has been detected. An acknowledgement with the unsent packet number has been ignored while according to the specification [13], these packets have to be declared as ERROR packets and the corresponding message has to be sent.

When testing **atfdp** another mismatching has been detected. When the acknowledgement with any number has been applied the server responded with a DATA packet with the next number. In other words, for example, after applying an input sequence (PRQ,0)(ACK_3,0) the output sequence DATA_1.DATA_4 has been observed which does not match the protocol specification [13].

IV. Conclusions

In this paper, we have studied a test suite that is derived based on the TFSM specification that is extracted from the TFTP description. The test suite has been derived as a transition tour of a corresponding FSM. As the performed experiments show, such test suites can be efficiently used for conformance testing of protocol implementations. A similar approach has been used for testing IRC and SMTP implementations [14, 15] and a number of inconsistencies have been detected. As a future work, we are going to study the fault coverage of test suites which are derived using other 'black box' testing methods.

References

[1] M. P. Vasilevskii, "Failure diagnosis of automata," translated from Kibernetika, No.4, 1973, pp. 98-108.

[2] T. S. Chow, "Test design modeled by finite-state machines," IEEE Trans. SE, vol. 4, No. 3, 1978, pp. 178-187.

[3] M. Dorofeeva, K. El-Fakih, S. Maag, A.R. Cavalli, N. Yevtushenko. FSM-based conformance testing methods: A survey annotated with experimental evaluation. Information and Software Technology, 52, 2010, pp. 1286-1297.

[4] J. Springintveld, F. Vaandrager, P. D'Argenio,.Testing Timed Automata. Theoretical Computer Science, 254(1-2), 2001, pp. 225-257.

[5] A. En-Nouaary, R. Dssouli, F.Khendek. Timed Wp-Method: Testing Real-Time Systems, IEEE TSE 28(11), 2002, pp. 1023-1038.

[6] K. El-Fakih, N. Yevtushenko, H. Fouchal. Testing Timed Finite State Machines with Guaranteed Fault Coverage. TestCom/FATES, 2009, pp. 66-80.

[7] M. G. Merayo, M. Nunez, I. Rodriguez. Extending EFSMs to Specify and Test Timed Systems with Action Durations and Time-outs. IEEE Transactions on Computers, 57(6), 2008, pp. 835-844.

[8] M. G. Merayo, M. Nunez, I. Rodriguez. Formal testing from timed finite state machines // Computer Networks. 52 (2), 2008, pp 432-460.

[9] M. Gromov, D. Popov, N. Yevtushenko. Deriving test suites for timed Finite State Machines. Proceedings of IEEE East-West Design & Test Symposium'08.,2008, pp. 339-343.

[10] M. Gromov, K. El-Fakih, N. Shabaldina, N. Yevtushenko. Distinguishing Non-deterministic Timed Finite State Machines, 11th Formal Methods for Open Object-Based Distributed Systems and 29th Formal Techniques for Networked and Distributed Systems, FMOODS/FORTE, LNCS 5522, 2009, pp. 137-151.

[11] M. Gromov, N. Yevtushenko. Synthesis of distinguishing test cases for timed finite state machines // Programming and Computer Software, 2010. – V. 36. – №.4. – P. 216-224.

[12] M. Zhigulin, N. Yevtushenko, S. Maag, A. Cavalli. FSM-Based Test Derivation Strategies for Systems with Time-Outs // 11th International Conference On Quality Software – Madrid, July 2011. – P. 141-150.

[13] RFC1350 – The TFTP Protocol (revision 2). URL: http://www.rfc-editor.org/rfc/rfc1350.txt

[14] M.V. Zhigulin, A.V. Kolomeez, N.G. Kushik, A.V. Shabaldin. Testing IRC implemintations based on extended finite state machine // Bulletin of the Tomsk Polytechnic University. – 2011. – V. 318, № 5. – P. 81-84 (in Russian).

[15] A. Nikitin, N. Kushik. On EFSM-based Test Derivation Strategies // Proceedings of the 4th Spring/Summer Young Researchers' Colloquium on Software Engineering. – Nizhny Novgorod, Russia. – 2010. – P. 116-119.

Formalization of Initial Requirements for the Design of Wireless Sensor Networks

Kislyakov, M.A.
Vladimir State University named after Alexander and Nikolay Stoletovs, VSU
Vladimir, Russian Federation
kislyakov.maxim@gmail.com

Mosin, S.G.
Vladimir State University named after Alexander and Nikolay Stoletovs, VSU
Vladimir, Russian Federation
smosin@ieee.org

***Abstract* — The paper discusses methods for the formal representation of the initial requirements for wireless sensor networks at the design stage. The main objective of the formalization is generating the basic mathematical tool and optimization criteria for individual stages of the design process. The automation of the design procedures with the use of computer technology is planned in terms of the proposed basis.**

Keywords – wireless sensor network; formalization; automation; optimization criterion

I. INTRODUCTION

Wireless sensor networks (WSN) are among the most relevant technologies at present. The applications of this technology are systematized, classified and include such basic directions as military defense, agriculture, industry, medicine and household sector [2, 3]. Moreover, the significance of the systems based on sensor networks is high in developed as well as in developing countries [1].

Currently, there are several manufacturers who offer their solutions on the market. Such solutions have a number of technical parameters which determine the range of use. It should be noted that the existing hardware and software complexes of sensor networks are created with the use of computer technology, but only individual stages of the design process are subject to automation. Consequently, the complex automation of the design procedure set is a relevant issue.

The task of automating the WSN design process involves the generation of the flow to be performed by the system under the supervision of a developer. In other words, the automation system is a software implementation of the design flow, where the developer is only a user.

The design of complex objects is based on the application of ideas and principles presented in a number of theories and approaches. The most common one is the *system approach* which determines the number of automation system components [4]. Firstly, the system must have a hierarchical structure. The hierarchical representation allows identifying a number of subsystems that can be designed independently. Secondly, the system must include analysis and simulation procedures. The purpose of these procedures is to identify a set of object parameters which will allow deciding on the next stage. Thirdly, the system must contain synthesis and optimization procedures which solve the problems of generating and modifying the object in order to achieve the parameters meeting the optimization criteria.

The automation design system allows for the development of the target object, but this is impossible without the formation of fixed criteria. Such criteria may be formed only on the basis of a number of a developer's initial requirements to the designed object. However, such requirements have a user-oriented format that is not suitable for the system. Therefore, the formalization of initial parameters is necessary.

The paper presents methods for the formalization of the initial requirements to a designed WSN object. The structure of the paper includes four sections. Section II contains a brief description of the design flow of WSN, according to which the optimization criteria based on the initial requirements for an object are proposed. Five key parameters that must be entered into the system by the user are identified in Section III. The proposed number of parameters is considered as sufficient to form a complete network object. Section IV concludes the paper.

II. DESIGN FLOW OF WSN

A detailed description of the design flow of WSN is not the purpose of this paper, but the general idea of the design process is necessary.

The automation design system is primarily determined by its flow. The flow determines the order of the design procedures and all the transitions between them. Consider the sequence of procedures applicable to WSN design.

The design flow of WSN comprises the following steps:

- Entering the initial requirements for WSN;
- Determination of a component basis;
- The synthesis of the basic structure;
- The use of the strategy for reliability;
- The use of the strategy for energy efficiency;
- Simulation;
- Generating the final project.

The stages of providing reliability and energy efficiency are iterative, which determines the presence of synthesis procedures as well as analysis procedures in the form of components. The design flow is also presented as a cycle, which ensures the process of parameter convergence of the designed object to the initial requirements.

III. Formalization of Initial Requirements

Entering the initial requirements of the project is the first procedure to be performed. The automation of entering the information is difficult. It is based on the full interaction of the developer and the system. However, the formalization of the input data can be automated with the use of mathematical tool proposed in the paper.

Five main parameters necessary for the design of WSN must be singled out of the set of initial requirements. The parameter list is given below:

- Life cycle of the network;
- Coverage area;
- Degree of information relevance;
- Data transmission rate;
- Level of network reliability.

The initial requirements are the basis for generating the internal parameters of the system. Their formalization is given below.

A. Life cycle of the network

The life cycle of the network determines the maximum uptime of the designed object. Theoretical calculation of the uptime of the network is presented in (1).

$$T_c = \min(T_y) \tag{1}$$

where T_c is the total uptime of the network, T_y is the set of uptime values of the nodes such that $T_y = \{t_{y1}, t_{y2}, \dots, t_{yn}\}$, n is the number of network nodes.

On the other hand, the life cycle of the node corresponds to the number of periods in the schedule (2).

$$t_{yi} = m_i t_{yp} \tag{2}$$

where $i = \{1..n\}$ is the index of the node, t_{yp} is the duration of one period in accordance with the schedule, m_i is the number of periods for the i-th node.

Five components of t_{yp} must be distinguished:

- t_{yp0} – sleep period;
- t_{yp1} – waiting period;
- t_{yp2} – period of reading sensor data;
- t_{yp3} – period of data reception from another node;
- t_{yp4} – period of data transmission to another node;
- t_{yp5} – period of information processing.

Parameter m_j matches period t_{ypj}. This parameter specifies the number of repetitions of period t_{ypj} for the node. Equation (3) is the calculation of period t_{yp}.

$$t_{yp} = \sum_{j=0}^{5} (m_j t_{ypj}) \tag{3}$$

Each period t_{ypj} is characterized by its indicator of energy consumption. Thus, parameter S_j which determines the discharge rate of the independent power supply unit corresponds to period t_{ypj}. In turn, the power supply has a certain capacity of P. Let the capacity of power supply for all nodes be equal to P. Then the optimization criterion has the form represented by (4).

$$P \geq \sum_{j=0}^{5} (m_j t_{ypj} S_j) \tag{4}$$

The values of parameters t_{ypj} and S_j are known. Parameter m_j is calculated based on the simulation of the object. Based on the initial formulation of the problem, the number of schedule periods m corresponds to the number of sleep periods m_0 of the node. Thus, the parameters of the life cycle of each node in the network can be calculated based on (2) and (4). The total uptime of the network is formed based on (1). Comparing the calculated parameter T_c with the initial requirements for the life cycle of the network, the system selects the next stage of the design flow.

B. Coverage area

The coverage area determines the spatial arrangement of network nodes which perform the role of sensors. Considering it, the network has a static topology, and location of nodes does not change with time. Then the network coverage can be represented as three sets: $X = \{x_1, \dots, x_n\}$, $Y = \{y_1, \dots, y_n\}$, $Z = \{z_1, \dots, z_n\}$.

Each node contains a transmitter with a certain power level. The power level of the transmitter determines the node action radius R in open space. Let the action radius for all nodes be equal to R.

Consider the case where all nodes can act as relays of information. Then the distance between nodes i and j in the three-dimensional space will be calculated by (5).

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{5}$$

where $i, j = (1..n)$ and $i \neq j$.

If the nodes connected to the sensors can not receive the information via radio channel, the relay must be added to the network. In this case, the calculation of the distance between the nodes is performed by (6).

$$D_{ij} = \sqrt{(x_i - x_j')^2 + (y_i - y_j')^2 + (z_i - z_j')^2} \tag{6}$$

where $i = (1..n)$, $j = (1..l)$, l is the number of relays added to the network, (x_j', y_j', z_j') are coordinates of the j-th relay. It

should be noted that the relays can be added in the case of equivalent nodes.

To fulfill the initial requirements of the coverage the system should ensure network connectivity. Each node must have at least one route for the transmission of information over the network. This requirement can be expressed by (7) and (8) for equivalent and nonequivalent nodes, respectively.

$$Q_i = \sum_{j=0}^{n} f(R \geq D_{ij}) \tag{7}$$

$$Q_i = \sum_{j=0}^{l} f(R \geq D_{ij}) \tag{8}$$

where $f(x)$ is the function of condition calculation. If the condition is satisfied, then $f(x) = 1$, otherwise $f(x) = 0$. Thus, the network connectivity is provided, if set Q contains no elements equal to zero. If the connectivity condition is not satisfied, an extra relay is added to the network.

C. Degree of information relevance

The transmission time from the source to the receiver determines the degree of information relevance. It is considered that for a period of scheduling a network node can receive and transmit no more than one information packet.

Let $M = \{m_1, \ldots, m_n\}$ be the set which defines the same route for each node in the network. For each route m_i parameter h_i is the dimension. The route dimension is numerically equal to the number of relays on the path from the source to the destination node.

Parameter T_a is the initial requirement to the degree of information relevance. This parameter is numerically equal to the maximum information transmission time. Then the relevance criterion can be determined by (9).

$$T_a \geq \max(H) \cdot t_{yp} \tag{9}$$

where $H = \{h_1, \ldots, h_n\}$ is a set of route dimensions.

If (9) is fulfilled, the relevance is provided, otherwise generating new routes is necessary.

D. Data transmission rate

The user forms the requirements to data transmission rate S_p in accordance with the target function of the network. In general terms, the user specifies the maximum transmission rate of each sensor node. The initial parameters are limited only by the bandwidth of the radio channel. However, the routes may overlap using shared relays. Thus, the bandwidth is reduced in proportion to the number of overlapping routes.

The bandwidth of the radio channel is defined as S_r. The next step is to define set $U = \{u_1, \ldots, u_n\}$, where u_i – the number of the routes which include the i-th network node. Then the initial requirement criterion for the data transfer rate is expressed by (10).

$$S_p \leq S_r / \max(U) \tag{10}$$

If (10) is fulfilled, the transfer rate is provided, otherwise generating new routes is necessary.

It should be noted that the parameters of relevance and data transmission rate are interrelated. Set $H_m = \{h_{m1}, \ldots, h_{mn}\}$ is added. Its elements are calculated according to (11). Then (9) takes the form of (12).

$$h_{mi} = h_i \cdot u_i \tag{11}$$

$$T_a \geq \max(H_m) \cdot t_{yp} \tag{12}$$

Consequently, the parameters of relevance and data transmission rate depend on set M. Reducing the number of intersections of the routes increases the data transmission rate and reduces the total time of message delivery.

E. Level of network reliability

The reliability of the network is one of its most important parameters. The failure probability of the network is defined as P_c. The failure probability of the node is defined as P_y. Parameter P_y is determined by the selected component basis. It is formed on the basis of specifications.

When generating a static network topology the routes can be calculated at the design stage. If each node doesn't use more than one route, then $P_c = P_y$. If the system provides a number of reserved routes, the value of P_c is reduced, but the exact mathematical representation is quite difficult to make. The simplest way to assess the reliability of the process will be running a simulation considering P_y and iterative analysis of the network connectivity. The reliability is supposed to be estimated based on statistical data.

IV. CONCLUSION

The methods for the formal representation of the initial requirements for WSN have been proposed in the paper. Mathematical models of optimization criteria which ensure the convergence of the design process have been examined. The proposed mathematical tool is aimed at automation of the project procedures in accordance with the design flow of WSN.

The development of CAD tools for WSN design based on the proposed mathematical models is the next task of the authors.

REFERENCES

[1] S. Taruna, Kusum Jain, G.N. Purohit, "Application Domain of Wireless Sensor Network: - A Paradigm in Developed and Developing Counties," IJCSI International Journal of Computer Science Issues, vol. 8, issue 4, no 2, July 2011, pp. 611–617.

[2] I. Khemapech, I. Duncan, A. Miller, "A Survey of Wireless Sensor Networks Technology," in PGNET, Proceedings of the 6th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking & Broadcasting, (Liverpool, UK), EPSRC, June 2005.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, "Wireless Sensor Networks: a Survey," Computer Networks, vol. 38, 2002, pp. 393–422.

[4] I.P. Norenkov, V.B. Manichev, "Osnovi teorii proektirovaniya SAPR [Basic Theory and Design CAD]," Moscow: High school, 1990, p. 335. (rus).

On Temporal Properties of Nested Petri Nets

Leonid Dvoryansky
Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
leo@mathtech.ru

Daniil Frumin
Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
difrumin@edu.hse.ru

***Abstract*—Nested Petri nets is an extension of Petri net formalism with net tokens for modelling multi-agent distributed systems with complex structure. Temporal logics, such as CTL, are used to state requirements of software systems behaviour. However, in the case of nested Petri nets models, CTL is not expressive enough for specification of system behaviour. In this paper we propose an extension of CTL with a new modality for specifying agents behavior. We define syntax and formal semantics for our logic, and give small examples of its usage.**

***Index Terms*—Petri nets, nested Petri nets, temporal logic, CTL**

I. Introduction

Petri nets is a popular formalism for modelling concurrent systems. Different extensions of Petri nets are extensively studied in the literature. The most popular are coloured Petri nets [5]. Nested Petri nets [6] is a formalism for modelling hierarchical multi-agent systems. There is a variety of temporal logics for specifying behavioural properties of discrete systems, such as HML, CTL, LTL, μ-calculi [4], [1], [3]. However, they are not convenient for expressing some natural properties of nested Petri nets.

The paper is organized as follows. To start with, we give some necessary foundations of labelled transition systems and Petri nets. Then we describe nested the Petri nets formalism. After that we give some examples of nested Petri nets properties we would like to express, and define nCTL – an extension of CTL for nested Petri nets. Finally we describe a formal semantics for nCTL. The paper ends with a conclusion.

II. Background

Definition 1. *A* Labelled Transition System (LTS) *is a tuple* (S, q_0, R, Act) *where*

- S *– a set of* states (worlds)*;*
- Act *– a set of actions;*
- $q_0 \in S$ *is an initial state;*
- $R \subseteq S \times Act \times S$ *is a transition relation.*

For convenience we write $s \xrightarrow{a} s'$ instead of $(s, a, s') \in R$.

Definition 2. *A* Petri net (P/T-net) *is a 4-tuple* (P, T, F, W) *where*

- P *and* T *are disjoint finite sets of* places *and* transitions*, respectively;*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a set of* arcs*;*
- $W : F \to \mathbb{N} \setminus 0$ *– an* arc multiplicity function*, that is, a function which assigns every arc a positive integer called an* arc multiplicity.

We denote by $\widetilde{W}$ the extension of W by zero

$$\widetilde{W}(x, y) = \begin{cases} n, & xFy \wedge W(x, y) = n \\ 0, & \neg xFy \end{cases}$$

A *marking* of a Petri net (P, T, F, W) is a multiset over P, i.e. a mapping $M : P \to \mathbb{N}$. By $\mathcal{M}(N)$ we denote a set of all markings of a P/T-net N.

We say that the transition t in a P/T-net $N = (P, T, F, W)$ is *active* in the marking M iff for every $p \in \{p \mid (p, t) \in F\}$: $M(p) \geq \widetilde{W}(p, t)$. An active transition may *fire*, resulting in a marking M', such as for all $p \in P$: $M'(p) = M(p) - \widetilde{W}(p, t) + \widetilde{W}(t, p)$.

III. Nested Petri Nets

In this section we define *nested Petri nets* (*NP-nets*) [6]. For simplicity we consider here only two-level NP-nets, where net tokens are usual Petri nets.

Definition 3. *A* nested Petri net *is a tuple* $(Atom, \mathit{Expr}, \mathit{Lab}, SN, (EN_1, \ldots, EN_k))$ *where*

- $Atom = \mathit{Var} \cup \mathit{Con}$ *– a set of atoms;*
- Lab *is a set of transition labels;*
- $(EN_1, \ldots, EN_k)$*, where* $k \geq 1$ *– a finite collection of P/T-nets, called* element nets*;*
- $SN = (P_{SN}, T_{SN}, F_{SN}, \upsilon, W, \Lambda)$ *is a high-level Petri net where*
  - P_{SN} *and* T_{SN} *are disjoint finite sets of* system places *and* system transitions *respectively;*
  - $F_{SN} \subseteq (P_{SN} \times T_{SN}) \cup (T_{SN} \times P_{SN})$ *is a set of* arcs*;*
  - $\upsilon : P_{SN} \to \{EN_1, \ldots, EN_k\} \cup \{\bullet\}$ *is a* place typing *function;*
  - $W : F_{SN} \to \mathit{Expr}$ *is an* arc labelling *function;*
  - $\Lambda : T_{SN} \to \mathit{Lab} \cup \{\tau\}$ *is a* transition labelling *function,* τ *is the special "silent" label;*

The arc expression language *Expr* is defined as follows. Let *Con* be a set of *constants* interpreted over $A = A_{net} \cup \{\bullet\}$ and $A_{net} = \{(EN, m) \mid \exists i = 1, \ldots, k : EN = EN_i, m \in \mathcal{M}(EN_i)\}$, i.e. A_{net} is a set of marked element nets. Let

The research is partially supported by the Russian Fund for Basic Research (project 11-01-00737).

Var be a set of *variables*. Then an expression in *Expr* is a multiset over *Con* $\cup$ *Var*. The arc labeling function W is restricted in such way that constants or multiple instances of the same variable are not allowed in input arc expressions of the transition, constants and variables in the output arc expressions should correspond to the types of output places, and each variable in an output arc expression of the transition should occur in one of the input arc expressions of the transition.

A marking M in a NP-net *NPN* is a function mapping each $p \in P_{SN}$ to some (possibly empty) multiset $M(p)$ over A. By abuse of notation, a set of all markings in a NP-net *NPN* is denoted by $\mathcal{M}(NPN)$.

A behavior of an NP-net consists of three kinds of steps. A *system-autonomous step* (resp. *element-autonomous step*) is a firing of a transition, labeled with τ, in the system net (resp. in one of the element nets).

An element-autonomous step is a transition firing according to the standard firing rules for P/T-nets.

To describe a system-autonomous step we need the concept of *binding*.

Definition 4. *Let Vars*(e) *denote a set of variables in an expression* $e \in$ *Expr. For each* $t \in T_{SN}$ *we define* $W(t) = \{W(x,y) \mid (x,y) \in F_{SN} \wedge (x = t \vee y = t)\}$ *– all expressions labelling arcs incident to* t*.*

A binding b *of a transition* t *is a function* b : *Vars*$(W(t)) \to A$*, mapping every variable in the* t*-incident arc expression to some value.*

We say that a transition t is *active* w.r.t. a binding b iff $\forall p \in \{p \mid (p,t) \in F_{SN}\}$: $b(W(p,t)) \subseteq M(p)$. An active transition may *fire* (denote $M \xrightarrow{t[b]} M'$) yielding a new marking $M'(p) = M(p) - b(W(p,t)) + b(W(t,p))$ for each $p \in P_{SN}$. An autonomous step in a net token changes only this token inner marking. An autonomous step in a system net can move, copy, generate, or remove tokens involved in the step, but doesn't change their inner markings.

A *(vertical) synchronization step* is a simultaneous firing of a transition, labeled with some $\lambda \in$ *Lab*, in a system net together with firings of transitions, also labeled with λ, in all net tokens involved in (i.e. consumed by) this system net transition firing. For further details see [6]. Note, however, that here we consider a typed variant of NP-nets, when a type of an element net is instantiated to each place.

IV. Temporal Logics

In this section we describe *CTL – computational tree logic*, which is widely used for specifying temporal properties of reactive systems.

A CTL formula is defined by the following grammar:
$\Phi ::= \mathsf{true} \mid (\neg\Phi) \mid (\Phi_1 \vee \Phi_2) \mid \mathbf{EU}(\Phi_1, \Phi_2) \mid \mathbf{AU}(\Phi_1, \Phi_2) \mid \mathbf{EX}(\Phi) \mid \mathsf{p}$
where p is an *atomic proposition*.

A CTL formula is interpreted over *Kripke structures*. Kripke structure is a labelled transition system (c.f. definition 1) where Act is a singleton $\{\tau\}$.

We can recursively define interpretation of a given CTL formula ϕ over a Kripke structure K and a current state s. We suppose some fixed interpretation of atomic propositions $I : S \times AP \to \{\text{true}, \text{false}\}$.

- $(K,s) \models \text{true}$;
- $(K,s) \models p$ iff $I(p,s) = \text{true}$;
- $(K,s) \models (\phi_1 \vee \phi_2)$ iff $(K,s) \models \phi_1$ or $(K,s) \models \phi_2$;
- $(K,s) \models (\neg\phi)$ iff $(K,s) \not\models \phi$;
- $(K,s) \models \mathbf{EX}(\phi)$ iff $\exists (s,s') \in R.\, (K,s') \models \phi$;
- $(K,s) \models \mathbf{EU}(\phi_1, \phi_2)$ iff there exists a *path* $s_1 s_2 \cdots$ in K (that is $s_1 \xrightarrow{\tau} s_2$, $s_2 \xrightarrow{\tau} s_3$, $\cdots$) such that: $s_1 = s$ and $\exists n.\, (\forall j \in \overline{0, n-1}.(K, s_j) \models \phi_1) \wedge (K, s_n) \models \phi_2$;
- $(K,s) \models \mathbf{AU}(\phi_1, \phi_2)$ iff for every path $s_1 s_2 \cdots$ in K the following holds: $s_1 = s$ and $\exists n.\, (\forall j \in \overline{0, n-1}.(K, s_j) \models \phi_1) \wedge (K, s_n) \models \phi_2$.

We can also define additional useful operator *weak until*: $\mathbf{AW}(\phi,\psi) = \neg\mathbf{EU}(\phi \wedge \neg\psi, \neg\phi \wedge \neg\psi)$, $\mathbf{EW}(\phi,\psi) = \neg\mathbf{AU}(\phi \wedge \neg\psi, \neg\phi \wedge \neg\psi)$. Intuitively, if model satisfies $\mathbf{AW}(\phi,\psi)$ (resp. $\mathbf{EW}(\phi,\psi)$) then for all paths (resp. there exists a path) in which either ϕ is true until we encounuter ψ or ϕ is always true. The difference between $\mathbf{AW}(\psi,\phi)$ and $\mathbf{AU}(\psi,\phi)$ is that in the former it's not necessary that ϕ is reached.

V. Temporal properties of NP-nets

Let's consider the following example (Fig.1). Here the left net is a system net with the net token α residing in p_1, and α depicted in the right part of Fig.1.

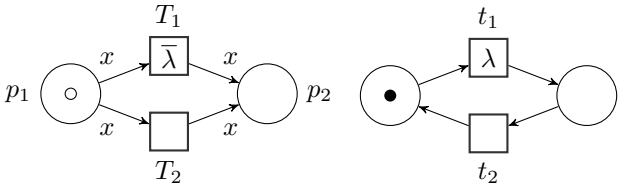


Fig. 1. NP-net NPN_1

LTS representing the behaviour of this NP-net is shown in Fig.2.

Imagine, we want to check that in every net token transitions t_1 and t_2 fire by turns. We could try the formula $\mathbf{AG}((t_1 \implies \mathbf{AXAW}(\neg t_1, t_2)) \wedge (t_2 \implies \mathbf{AXAW}(\neg t_2, t_1)))$. Although this approach might look attractive at the first sight, it does not work in many cases.

To show why our approach does not work here let's take a closer look at the LTS (Fig.4) corresponding to NPN_2 (Fig.4).

Since t_1 appears "before" t_2 in LTS Σ_2, we can conclude that NPN_2 satisfies our formula. However, actually in the second net token t_2 fired before t_1. The problem is that transition firings in different net tokens are indistinguishable in our model: t_2 in figure 4 refers to the firing of t_2 in the net token which originally resided in q_3, but Σ_2 does not contain any information about that.

To handle such properties we introduce a new modality in the next section.

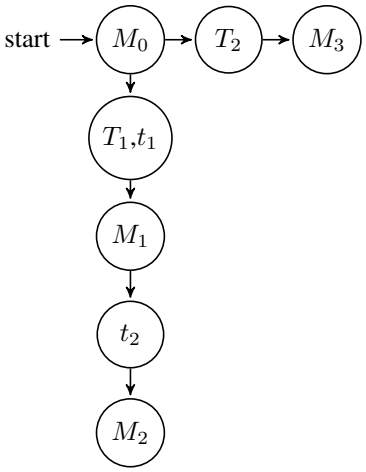


Fig. 2. LTS corresponding to *NPN*$_1$

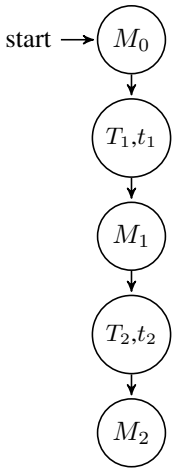


Fig. 4. LTS Σ_2 of *NPN*$_2$

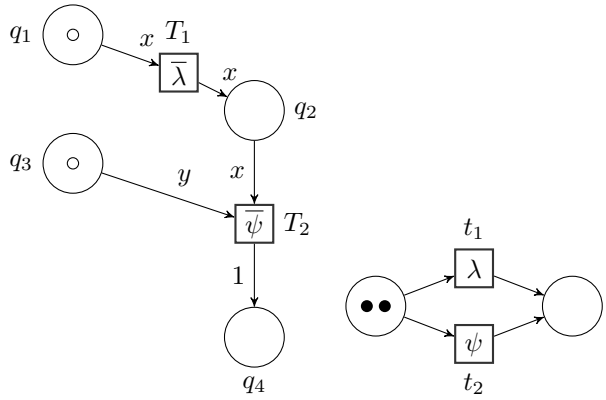


Fig. 3. NP-net *NPN*$_2$

VI. nCTL

In the following section we present a solution to the problem described in the previous passage by introducing additional modality.

A. Syntax

To specify properties concerning states (markings), system transitions and transitions in element nets we introduce a logic with three categories of formulae. Just like in CTL we make use of path quantifiers in both state and transition formulae.

We define the syntax of nCTL with a fixed nested Petri net *NPN* in mind.

State formulae: $\mathcal{A} ::= \mathsf{true} \mid \omega \mid \neg\mathcal{A} \mid (\mathcal{A}_1 \vee \mathcal{A}_2) \mid \langle\mathcal{B}\rangle \mid [\mathcal{C}] \mid \mathbf{EU}(\mathcal{A}_1, \mathcal{A}_2) \mid \mathbf{AU}(\mathcal{A}_1, \mathcal{A}_2)$

Transition formulae: $\mathcal{B} ::= \mathsf{true} \mid \chi \mid \neg\mathcal{B} \mid (\mathcal{B}_1 \vee \mathcal{B}_2) \mid \langle\mathcal{A}\rangle \mid \mathbf{EU}(\mathcal{B}_1, \mathcal{B}_2) \mid \mathbf{AU}(\mathcal{B}_1, \mathcal{B}_2)$

Element transition formulae: $\mathcal{C} ::= \mathsf{true} \mid \gamma \mid \neg\mathcal{C} \mid (\mathcal{C}_1 \vee \mathcal{C}_2) \mid [\mathcal{A}] \mid \mathbf{EU}(\mathcal{C}_1, \mathcal{C}_2) \mid \mathbf{AU}(\mathcal{C}_1, \mathcal{C}_2) \mid \mathbf{AX}\mathcal{C}$

Here true is a boolean constant, ω is a function, called a *marking predicate*, with the type $\mathcal{M} \to \{\text{true}, \text{false}\}$, i.e. a predicate on the set of all markings. A function χ maps transitions $\mathcal{T}$ of SN to booleans. γ is a predicate on set of all transitions of all element nets (we do not need a predicate on markings of element nets, since every marking of an element net can be characterized by a subset of $\mathcal{M}$). $\mathbf{EU}$ and $\mathbf{AU}$ are familiar from the conventional CTL.

A nCTL formula is a well-formed state formula.

B. Examples

The idea of using both state and event modalities were first developed in ASK-CTL library for coloured Petri nets [2]. We extend this idea to NP-nets.

Intuitively, when we encounter an element transition subformula, we switch our interpreting context to an LTS of an element net. Nesting of the modalities allow us to switch back and forth between contexts. $[\phi]$ means that there exists a path in the LTS of the NP-net along which ϕ holds for every element net.

Now we can express the "switching" property for the *NPN*$_2$ (Fig.3): $[\mathbf{AU}(\neg t_2, t_1) \wedge \mathbf{AG}(t_1 \implies \mathbf{AXAU}(\neg t_1, t_2)) \wedge \mathbf{AG}(t_1 \implies \mathbf{AXAU}(\neg t_1, t_2))]$. The $\mathbf{AU}(\neg t_2, t_1)$ part is necessary to check whether t_1 is the first transition to be fired.

In order to properly verify whether the LTS, corresponding to *NPN*$_2$, is a model for our formula, we should change the way we construct the LTS. Firsly, we introduce a set $N = \{n_0, n_1, \dots\}$ every member of which represents a single element net token (note that N is different from A_{net}, since the latter contains only types of element nets together with their markings, while the former also distinguishes between individual tokens). Now we mark arcs in a LTS with tuples of the form (τ, n_i), where τ a name of transition in an element token $n_i \in N$. From here on in we use notation $\tau[n_i]$ to denote (τ, n_i).

The new LTS corresponding to the *NPN*$_2$ is shown in Fig.5. The element net tokens residing in q_1 (resp. q_3) is denoted as n_0 (resp. n_1). If we check the only path generated by transitions – $(T_1, t_1[n_0])(T_2, t_2[n_1], t_2[n_0])$ – we see that it does not satisfy our formula.

It is worth mentioning that our approach is not equivalent to model checking of element transition formulae on LTSs corresponding to element nets. This is caused by the fact

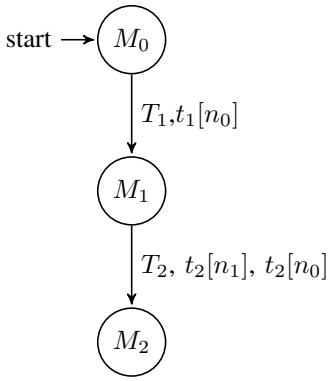


Fig. 5. New LTS of *NPN*$_2$

that the LTS corresponding to an element net should be considered only w.r.t. transitions in the system net due to vertical synchronization. In addition, nCTL provides an ability to switch back to system context, based on the properties of an element net. Consider this example: $[(\mathbf{AX}t_1 \implies [M_1]) \vee (\mathbf{AX}t_2 \implies [\langle \mathbf{EX}T_2 \rangle])]$. NP-net N' models that formula if either t_1 fires in the next step in the element net and N' reaches marking M_1 or t_2 fires in the next step in the element net and T_2 is enabled in the system net after that.

We need to construct LTS with respect to information about specific element nets. Let us consider some concrete problems conected with that.

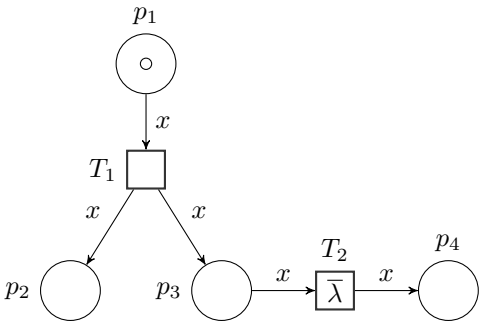


Fig. 6. NP-net *NPN*$_3$

Fig.6 represents an example of a nested Petri net, where the firing of transition T_1 yields two copies (n_0 in p_2, n_1 in p_3) of the same element net, therefore in both copies we need to keep the history of the old net token. For example, if the net token κ (Fig.7) resides in the place p_1 in *NPN*$_3$, we get the LTS shown in Fig.8.

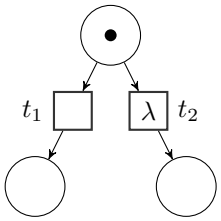


Fig. 7. Net token κ in *NPN*$_3$

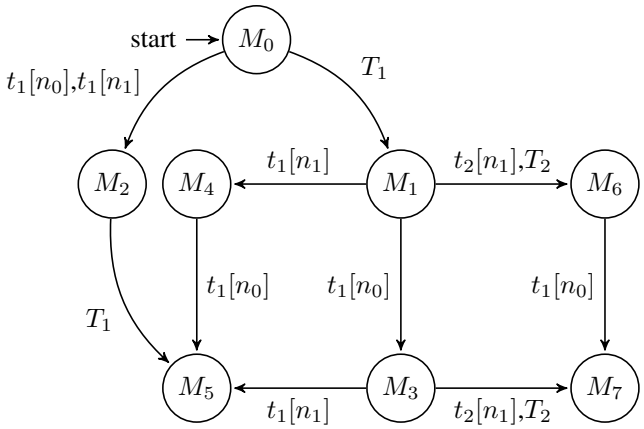


Fig. 8. LTS corresponding to *NPN*$_3$

VII. nCTL SEMANTICS

nCTL formulae are interpreted over a pair (L_{NPN}, m), where L_{NPN} is a LTS corresponding to *NPN* and m is a reachable marking of the latter. We say that *NPN* satisfies formula ϕ iff $(L_{NPN}, m_0) \models \phi$, where m_0 is the initial marking in *NPN*. State-transition modalities allow us to switch between different types of formulae.

Interpretation for state formulae:

- $(L_{NPN}, m) \models_{\mathcal{A}}$ true
- $(L_{NPN}, m) \models_{\mathcal{A}} \omega \iff \omega(m)$
- $(L_{NPN}, m) \models_{\mathcal{A}} \neg\phi \iff (L_{NPN}, m) \not\models_{\mathcal{A}} \phi$
- $(L_{NPN}, m) \models_{\mathcal{A}} \phi_1 \vee \phi_2 \iff (L_{NPN}, m) \models_{\mathcal{A}} \phi_1$ or $(L_{NPN}, m) \models_{\mathcal{A}} \phi_2$
- $(L_{NPN}, m) \models_{\mathcal{A}} \langle\phi\rangle \iff \exists a \,.\, m \xrightarrow{a} m' \wedge (L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \phi$
- $(L_{NPN}, m) \models_{\mathcal{A}} [\phi] \iff \exists a \,.\, m \xrightarrow{a} m' \wedge (L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \phi$
- $(L_{NPN}, m) \models_{\mathcal{A}} \mathbf{AU}(\phi_1, \phi_2) \iff \forall \sigma = a_1 a_2 \cdots \in \mathcal{P}_m \,.\, \exists n \leq |\sigma| \,.\, m \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n \,.\, (\forall i \in \overline{0, n-1} \,.\, (L_{NPN}, m_i) \models_{\mathcal{A}} \phi_1) \wedge (L_{NPN}, m_n) \models_{\mathcal{A}} \phi_2$
- $(L_{NPN}, m_0) \models_{\mathcal{A}} \mathbf{EU}(\phi_1, \phi_2) \iff \exists \sigma = a_1 a_2 \cdots \in \mathcal{P}_m \,.\, \exists n \leq |\sigma| \,.\, m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n \,.\, (\forall i \in \overline{0, n-1} \,.\, (L_{NPN}, m_i) \models_{\mathcal{A}} \phi_1) \wedge (L_{NPN}, m_n) \models_{\mathcal{A}} \phi_2$

Interpretation for transition formulae:

- $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}}$ true
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \chi \iff \bigwedge\{\chi(\tau) \mid \tau \in ST(a)\}$, where $ST(a)$ is a set of system transitions, involved in the step a.
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \neg\phi \iff (L_{NPN}, (m, a, m')) \not\models_{\mathcal{B}} \phi$
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \phi_1 \vee \phi_2 \iff (L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \phi_1$ or $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \phi_2$
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{B}} \langle\phi\rangle \iff (L_{NPN}, m') \models_{\mathcal{A}} \phi$
- $(L_{NPN}, (m_0, a_1, m')) \models_{\mathcal{B}} \mathbf{AU}(\phi_1, \phi_2) \iff \forall \sigma = a_1 a_2 \cdots \in \mathcal{P}_m \,.\, \exists n < |\sigma| \,.\, m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n \,.\, (\forall i \in$

$\overline{0, n-1}\,.\,(L_{NPN}, (m_i, a_i, m_{i+1}) \models_{\mathcal{B}} \phi_1) \wedge (L_{NPN}, (m_n, a_{n+1}, m_{n+1}) \models_{\mathcal{B}} \phi_2)$

- $(L_{NPN}, (m_0, a_1, m')) \models_{\mathcal{B}} \mathbf{EU}(\phi_1, \phi_2) \iff \exists \sigma = a_1 a_2 \cdots \in \mathcal{P}_m\,.\,\exists n < |\sigma|\,.\,m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n\,.\,(\forall i \in \overline{0, n-1}\,(L_{NPN}, (m_i, a_i, m_{i+1}) \models_{\mathcal{B}} \phi_1) \wedge (L_{NPN}, (m_n, a_{n+1}, m_{n+1}) \models_{\mathcal{B}} \phi_2)$

Interpretation for nested transition formulae:

- $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}}$ true
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \gamma \iff \bigwedge\{\gamma(\tau[n]) \mid \tau[n] \in NT(a)\}$, where $NT(a)$ is a set of element net transitions involved in a step a.
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \neg\phi \iff (L_{NPN}, (m, a, m')) \not\models_{\mathcal{C}} \phi$
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \phi_1 \vee \phi_2 \iff (L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \phi_1$ or $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}} \phi_2$
- $(L_{NPN}, (m, a, m')) \models_{\mathcal{C}} [\phi] \iff (L_{NPN}, m) \models_{\mathcal{A}} \phi$
- $(L_{NPN}, (m_0, a_1, m')) \models_{\mathcal{C}} \mathbf{AU}(\phi_1, \phi_2) \iff \forall \sigma = a_1 a_2 \cdots \in \mathcal{P}_m\,.\,\exists n < |\sigma|\,.\,m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n\,.\,(\forall i \in \overline{0, n-1}\,.\,(L_{NPN}, (m_i, a_i, m_{i+1}) \models_{\mathcal{C}} \phi_1) \wedge (L_{NPN}, (m_n, a_{n+1}, m_{n+1}) \models_{\mathcal{C}} \phi_2)$
- $(L_{NPN}, (m_0, a_1, m')) \models_{\mathcal{C}} \mathbf{EU}(\phi_1, \phi_2) \iff \exists \sigma = a_1 a_2 \cdots \in \mathcal{P}_m\,.\,\exists n < |\sigma|\,.\,m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots m_{n-1} \xrightarrow{a_n} m_n\,.\,(\forall i \in \overline{0, n-1}\,.\,(L_{NPN}, (m_i, a_i, m_{i+1}) \models_{\mathcal{C}} \phi_1) \wedge (L_{NPN}, (m_n, a_{n+1}, m_{n+1}) \models_{\mathcal{C}} \phi_2)$
- $(L_{NPN}, (m_0, a_1, m')) \models_{\mathcal{C}} \mathbf{AX}\phi \iff \forall \sigma = a_1 a_2 \cdots \in \mathcal{P}_m\,.\,(m_0 \xrightarrow{a_1} m_1 \xrightarrow{a_2} m_2 \ldots) \wedge (L_{NPN}, (m_1, a_2, m_2) \models_{\mathcal{C}} \phi)$

From a practical perspective, nCTL logic enables us to state properties of agents in multiagent systems.

VIII. Conclusion and future research

The temporal logic nCTL, described in this paper, is an extension of CTL for specifying semantic properties of nested Petri nets. Logic nCTL may be helpful for describing behavioural properties of multi-agent systems with complex structure. nCTL allows to express both system net and element nets properties directly. This gives a straightforward way of formalizing NP-net specific temporal properties.

Our next goal is to develop an algorithm for constructing LTS describing semantics of a given NP-net. We also plan to investigate the expressive power of nCTL and study the possibility of developing effective verification algorithm for nested Petri nets.

References

[1] Julian Bradfield, Colin Stirling, *Modal mu-calculi*, In: Patrick Blackburn, Johan Van Benthem and Frank Wolter, Editor(s), Studies in Logic and Practical Reasoning, Elsevier, 2007, Volume 3, Pages 721-756.

[2] A. Cheng, S. Christensen, and K. H. Mortensen, *Model Checking Coloured Petri Nets Exploiting Strongly Connected Components*. – Citeseer, 1997.

[3] Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*, MIT Press, 2001.

[4] Ian Hodkinson, Mark Reynolds, *Temporal logic*, In: Patrick Blackburn, Johan Van Benthem and Frank Wolter, Editor(s), Studies in Logic and Practical Reasoning, Elsevier, 2007, Volume 3, Pages 655-720.

[5] K. Jensen and L. M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer, 2009.

[6] I. A. Lomazova, *Nested Petri nets: Modeling and analysis of distributed systems with object structure*. – Moscow:Scientific World, 2004. – 208p.

Checking service compatibility via resource conformance

Ivan Romanov
Software Engineering School
National Research University Higher School of Economics
33/5 Kirpichnaya, st. Moscow, Russian Federation
Email: romanov.ekb@gmail.com

Scientific Advisor: Prof. Irina A. Lomazova
Software Engineering School
National Research University Higher School of Economics
33/5 Kirpichnaya, st. Moscow, Russian Federation
Email: ilomazova@hse.ru

***Abstract* — In this work we consider modeling of services with workflow modules, which are a subclass of Petri nets. Service compatibility problem is studied. Quasi-regular expressions are used for checking service compatibility via resource conformance.**

Keywords – service composition; service compatibility; workflow net; workflow module

I. INTRODUCTION

Service-Oriented Computing [1] is an emerging computing paradigm that supports the modular design of (software) systems. Complex systems are designed by composing less complex systems, called *services*.

A service consists of a *control structure* describing its behavior and of an *interface* to communicate asynchronously with other services. Component interaction necessitates a mechanism of communication between services. An interface is a set of (input and output) channels. In order that two services can interact with each other, an input channel of the one service has to be connected with an output channel of the other service

There are a lot of algorithms for checking services compatibility. They allow, for example, to check deadlock freedom property, termination property, and others. Very often there methods have high computational complexity, affecting the speed of their implementation. It is assumed, that for service execution resources consumed from external environment are needed. In this paper a property of service compatibility via resource conformance is checked in the initial stage, it may help to avoid further verification because of evidently incompatible services.

In this work we consider modelling of services with workflow modules, which is a special subclass of Petri nets. Service compatibility problem is studied. Quasi-regular expressions are used for service compatibility via resource conformance analysis.

II. FORMAL MODEL FOR SERVICES

While analyzing and verifying service compatibility, let us abstract from underlying technologies and implementations and consider formal model based on Petri nets.

Petri nets is a popular formalism for modeling and analysis of distributed systems. A Petri net $N = (P; T; F)$ consists of a set of transitions T, a set of places P, and a flow relation $F \subseteq (P \times T) \cup (T \times P)$. Graphically, a place is represented by a circle, a transition by a box, and the flow relation by directed arcs between them. Whilst transitions represent dynamic elements, for example, an activity of a service places represent static elements for example, a condition to perform an activity of a service. A state of a Petri net is represented by a marking, which is a distribution of tokens over the places. Graphically, a token is depicted by a black dot.

Workflow net (WF net) is a special Petri net. This formalism is used to model workflow systems [3]. A workflow net has one initial and one final place, and every place or transition in it is on a directed path from the initial to the final place.

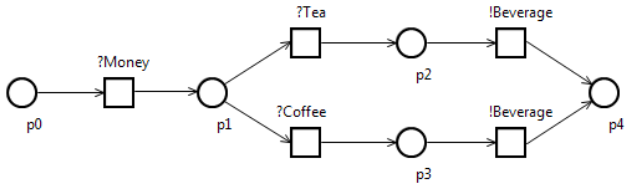


Figure 1. Example of workflow net. Model of a vending machine that sells, either a cup of tea, or coffee.

Modeling workflow [5] consists of modeling case management with the help of *parallelism*, *sequential routing*, *conditional routing* and *iteration*. To express it explicitly building blocks such as the AND-split, AND-join, OR-split and OR-join can be used. The AND-split and the AND-join are used for parallel routing. The OR-split and the OR-join are used for conditional routing. When we model an OR-split in terms of a Petri net, the OR-split corresponds to a number of transitions sharing the same set of input places. Other constructs also can be easily expressed in Petri net formalism.

To guarantee, that we get 'good' workflows, we are to balance AND/OR-splits and AND/OR-joins. Clearly, two parallel flows initiated by an AND-split, should not be joined by an OR-join. Two alternative flows created via an OR-split, should not be synchronized by an AND-join. When we follow these rules we obtain *structured WF nets* (see [3] for more

details). It is shown there, that structured WF nets are sound by construction.

A stateful service defines an internal process (i.e. activities building its internal structure), and an interface to communicate with other services. To model services we will use a special subclass of Petri nets called workflow module.

A Petri net $M = (P; T; F)$ is called workflow module [2],[4] if the following conditions hold:

1) The set of places is divided into three disjoint sets: internal places P^N, input places P^I and output places P^O.

2) The flow relation is divided into internal flow $F^N \subseteq (P^N \times T) \cup (T \times P^N)$ and communication flow $F^C \subseteq (P^I \times T) \cup (T \times P^O)$.

3) The net $PM = (P^N; T; F^N)$ is a workflow net.

4) No transition is connected both to an input place and an output place.

Within a workflow module M, the workflow net PM is called the internal process of M and the tuple $\mathcal{I}(M) = (P^I ; P^O)$ is called its interface.

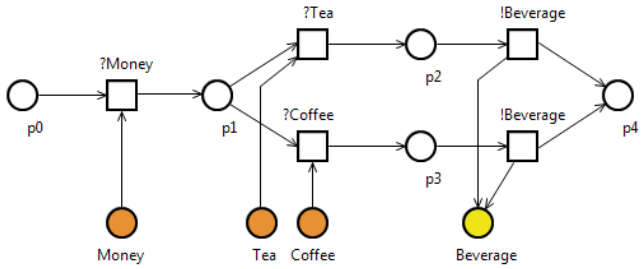


Figure 2. Example of a workflow module."Money","Tea","Coffee" are input places, "Beverage" is output place.

Denote structured workflow module (SWFM) as a workflow module, where the net $PM = (P^N; T; F^N)$ is a structured workflow net.

In this paper we consider the class of services, which can be modeled by SWFM.

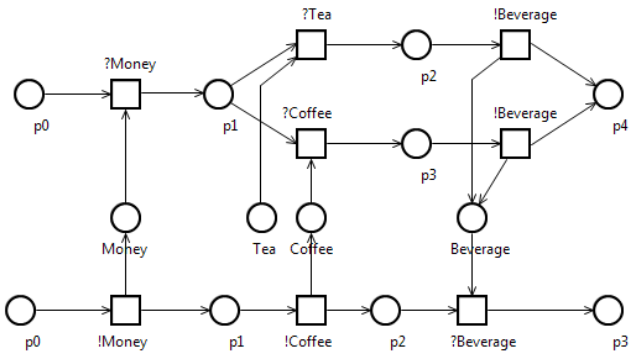


Figure 3. Example of service composition

Figure 3 illustrates an example of service composition. It shows situation when customer wants to buy coffee.

III. Service Compatibility via Resource Conformance

Let S be a finite set. A multiset m over a set S is a mapping m: S → ℕ, i.e. a multiset may contain several copies of the same element. By $\mathcal{M}(S)$ we denote the set of all finite multisets over S.

Let N is structured workflow module with two disjoint sets P^I of input places and P^O of output places. Consider some run δ, states sequence from initial to final state, of the N. For each run pair of input and output resources $(R_I; R_O)$,where $R_I \subseteq \mathcal{M}(P^I)$ and $R_O \subseteq \mathcal{M}(P^O)$ are defined. Denote $\rho(\delta) = (R_I; R_O)$ as resource of N [2].

For the considered class of services, let us denote quasi-regular expression defining a set of all possible resources for a service. For such type of resources, inner workflow net is structured, therefore, it becomes possible to identify the expression recursively by its structure:

1) Expression takes the form $R_I \circ R_O$ for atomic net where transition consumes R_I and produces R_O resources. Thus, rewrite this expression as

$$?r_1 \circ ?r_2 \circ \ldots \circ ?r_k \circ !r_{k+1} \circ !r_{k+2} \circ \ldots \circ !r_{k+m}$$

where $r_1, \ldots, r_k$ – input resources; $r_{k+1}, \ldots, r_{k+m}$ – output resources; If the transition does not produce/consume resources it is denoted as ε.

2) For sequential routing $e_1 \circ e_2$

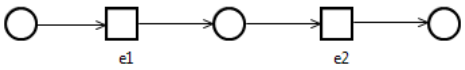


Figure 4. Sequential routing.

3) For parallel routing $e_1 \circ e_2$

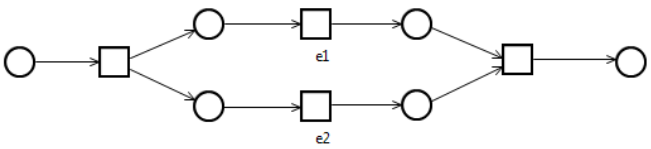


Figure 5. Parallel routing.

4) For selective routing $e_1 + e_2$

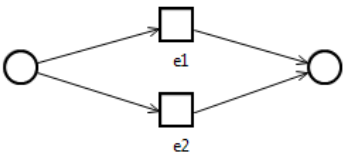


Figure 6. Selective routing.

5) For iteration $e_1 + (e_2 \circ e_1)^*$

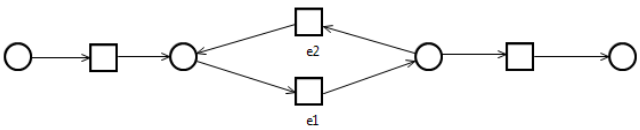


Figure 7. Iteration.

Proposition 1. Let expressions e1 and e2 define a set of resources for two structured workflow modules N_1 and N_2. Then after application of described operations deduced SWFM will possess the following resource sets: $e_1 \circ e_2$ for sequential, parallel routing, $e_1 + e_2$ for selective routing and $e_1 + (e_2 \circ e_1)^*$ for iteration.

Describe some properties of the algebra that we have:

1) $e \circ \varepsilon = e$
2) $\varepsilon^* = \varepsilon$
3) $e \circ e^* = e^+$
4) $(e^*)^* = e^*$
5) $e_1 \circ e_2 = e_2 \circ e_1$
6) $e_1 + e_2 = e_2 + e_1$
7) $e_1 \circ (e_2 + e_3) = e_1 \circ e_2 + e_1 \circ e_3$

For this class of quasi-regular expressions, as opposed to regular languages, a sequence of producing/consuming resources does not matter. That is why property 5 is true for quasi-regular expressions, while it is false for regular ones.

Using rules described, we may similarly compose a quasi-regular expression for the service investigated. To make representations more compact, we suppress interface conditions. Depicting a transition by a set of necessary resources is quite sufficient.

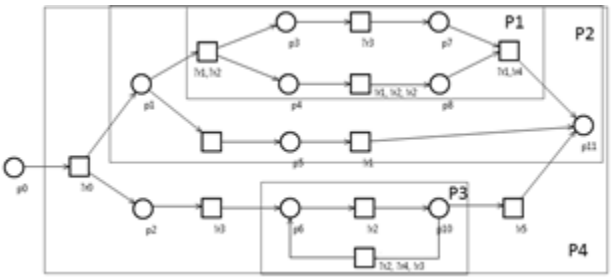


Figure 8. SWFN.

For example let us build quasi-regular expression for SWFN illustrated in Figure 8. We will consider resulted formula by parts P1, P2, P3, P4.

$$P1\colon (?r1 \circ ?r2 \circ (?r3 \circ !r1 \circ !r2 \circ !r2) \circ ?r1 \circ !r4)$$

$$P2\colon (!r1 + P1)$$

$$P3\colon !r2(?r2 \circ ?r4 \circ !r3 \circ !r2)$$

$$P4\colon ?r0 \circ (P2 \circ !\mathrm{r3} \circ \mathrm{P3} \circ !\mathrm{r5})$$

Expression for SWFN:

$$?r0 \circ ((!r1 + (?r1 \circ ?r2 \circ (?r3 \circ !r1 \circ !r2 \circ !r2) \circ ?r1 \circ !r4)) \circ !\mathrm{r3} \circ !r2(?r2 \circ ?r4 \circ !r3 \circ !r2) \circ !\mathrm{r5})$$

To check services equivalence and other properties, we need a unified representation for services. In order to achieve this, it is necessary to develop an algorithm for expressions reduction to normal form.

Two resources with equal representation, which is symmetric about '?' and '!' we denote as *complementary*. For example, $!\mathrm{r1} \circ ?\mathrm{r2} \circ !\mathrm{r3}$ and $?\mathrm{r1} \circ !\mathrm{r2} \circ ?\mathrm{r3}$ are complementary.

Two services we denote as *compatible via resource conformance* if their quasi-regular expressions contain two complementary resources.

Proposition 2. If service behaviors are compatible to each other, then two services are *compatible via resource conformance*.

Thus it is prerequisite condition for service compatibility.

IV. CONCLUSION

In this paper a service formal model is built and expression algebra for analyzing service compatibility via resource conformance, abstracting from underlying technologies and implementations is described. Detecting resource incompatibility in one of the first stages allows to avoid further verification consuming significant computing resources.

Further investigation in this field will be devoted to normal form construction for expressions, exploring new expression properties and compatibility proposition corroboration.

Besides, a piece of software is planned to be developed. Its main functions will concern service model and service expression construction, and compatibility checking by the properties described.

REFERENCES

[1] C. Stahl. Service Substitution – A Behavioral Approach Based on Petri Nets Dissertation, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II; Eindhoven University of Technology, December 2009.

[2] V. A. Bashkin and I. A. Lomazova. Resource equivalence in workflow nets Proc. CS&P'2006. Vol.1. Informatik-Bericht 206. Humboldt-Universitat zu Berlin. Berlin, 2006, pp. 80-91.

[3] W. van der Aalst and K. van Hee. Workflow Management: Models, Methods and Systems MIT Press, 2002.

[4] Axel Martens. Analyzing Web Service based Business Processes. In Maura Cerioli, editor, International Conference on Fundamental Approaches to Software Engineering (FASE 2005), volume 3442 of Lecture Notes in Computer Science, Edinburgh, Scotland, April 2005. Springer. [36, 40, 216, 217]

[5] I. A. Lomazova. Interacting Workflow Nets for Workflow Process Re-Engineering. Fundamenta Informaticae, 2010. T.101.№1-2. C.59—70

[6] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. In 2nd South-East European Workshop on Formal Methods 2005 (SEEFM05), Ohrid, Republic of Macedonia, 2005.

Elaborating on the alias calculus

Alexander Gerasimov
Saint Petersburg State University,
Computer Science Chair
and
Saint Petersburg National Research University of
Information Technologies, Mechanics and Optics,
Software Engineering Laboratory
Email: asgerasimov@gmail.com

***Abstract*—In this research-in-progress report, we elaborate on the alias calculus introduced in [1], [2]. The alias calculus is to determine whether two reference expressions at the given program point may address the same object at the run-time (in other words, whether one expression may be an alias of the other expression). The main intended application of the alias calculus is to support deductive verification of object-oriented programs. We show how aliases may be used in Hoare-style reasoning, hence derive at what program points we are to compute aliases and propose an algorithm that computes the required aliases. We also state future work directions.**

I. Introduction

To perform deductive verification of an object-oriented program we need to know whether two reference expressions at the given program point may address the same object at the run-time (in other words, whether one expression may be an alias of the other expression). For example, we should take care of such a situation. If x and y are aliased and an operation modifies the value of the attribute $x.f$, then the opertation also modifies $y.f$, though y is not mentioned in the text of the operation. The alias calculus introduced in [1], [2] is to give an answer on the formulated question.

[2] suggests to represent possible aliases at a program point as a symmetric and irreflexive relation over a set of reference expressions (such a relation is called an *alias relation*) and formulates rules of the alias calculus for a model of a programming language.

There are following instructions in the model of a programming language: skip, forget, create, assignment, compound, conditional, cut, loop, and procedure call.

Each rule is of the form

$$(a \gg p) \;=\; A,$$

where A denotes an alias relation that may hold just after the execution of a program instruction p provided an alias relation a holds just before the execution. Let us formulate some of the rules.

The rule for an assignment in the non-object-oriented alias calculus (where every reference expression is a variable) is

$$(a \gg x := y) \;=\; a[x : y],$$

where

- a is an alias relation, x and y are variables,
- $a[x : y] = b \cup \overline{(\{x\} \times (b/y))}$,
- $b = a \setminus\!\!- \{x\} \;=\; a \setminus \{\langle u, v\rangle \in a \mid u = x \vee v = x\}$,
- $b/y \;=\; \{y\} \cup \{u \mid \langle u, y\rangle \in b\}$,
- $\overline{c} = (c \cup \{\langle v, u\rangle \mid \langle u, v\rangle \in c\}) \setminus \{\langle u, u\rangle \mid \langle u, u\rangle \in c\}$ for a relation c.

The rule for a compound instruction is

$$(a \gg (p; q)) \;=\; (a \gg p) \gg q,$$

where p and q are instructions.

A conditional instruction in the model language is of the form "$\mathbf{then}\ p\ \mathbf{else}\ q$" (there is no boolean condition in it). The rule for it is

$$(a \gg \mathbf{then}\ p\ \mathbf{else}\ q) \;=\; (a \gg p) \cup (a \gg q).$$

The alias calculus may say that some expressions are aliased though in fact they cannot because the boolean condition in a real conditional and loop is not taken into account. Such an imprecision may be handled by means of a cut instruction in the model language. The instruction "$\mathbf{cut}\ e_1, e_2$" asserts that the expressions e_1 and e_2 are not aliased at the given program point. In the non-object-oriented alias calculus the rule for "$\mathbf{cut}\ e_1, e_2$" is

$$(a \gg \mathbf{cut}\ e_1, e_2) \;=\; a \setminus \overline{e_1, e_2},$$

where $\overline{s} = \overline{s \times s}$ for a set s of expressions and the braces enclosing a list of the elements of s may be omitted.[1]

A. Related work.

The following approaches to handling references for program verification are known (see [2], [3]): separation logic [4], shape analysis [5], ownership types [6], and dynamic frames [7]. However separation logic and shape analysis try to reveal a more detailed structure of pointers than it is necessary for alias analysis [2]. Separation logic, ownership types, and dynamic frames require a programmer to write additional annotations besides Hoare assertions.

The alias calculus is formulated in terms of a model of a high-level programming language and additional annotations (i. e., cut instructions) are rarely required.

[1] Thus $\overline{e_1, e_2} = \{\langle e_1, e_2\rangle, \langle e_2, e_1\rangle\}$. The notation $\overline{s}$ is also used below.

B. Our work.

We analyse the alias calculus [2] and its prototype implementation (mentioned in [2]) and elaborate on some aspects of the alias calculus in order to provide a solid basis for further implementation and integration of the alias calculus into Eiffel Verification Environment [8]. These aspects are:

- Hoare-style reasoning with aliases;
- computing aliases for calls to (mutually) recursive procedures;
- coping with the infinity of some alias relations in the object-oriented alias calculus.

These aspects and some of our results achieved are described in the subsequent sections.

II. On Hoare-style reasoning with aliases

Consider the following Eiffel code fragment:

---

if b **then** $y := x$
$--$ $true$
$x.set$
$--$ $x.f = c$

Suppose the precondition and postcondition of the qualified procedure call $x.set$ are given in the comments above (the comments are the lines starting with "$--$"). Let us try to infer the weakest precondition of this fragment for the postcondition $y.f = c$.

If we do not know that x and y may be equal just after the call $x.set$, then we cannot weaken the postcondition $x.f = c$ to $y.f = c$.

Suppose we have computed aliases that may hold just before and just after the call $x.set$: x and y may be aliased at these program points. Then we add $x = y$ to the precondition and postcondition of $x.set$ via conjunction. Now we obtain the precondition $b \vee x = y$ of the whole code fragment using Hoare rules [9]:

$--$ $b \vee x = y$
if b **then** $y := x$
$--$ $true \wedge x = y$
$x.set$
$--$ $x.f = c \wedge x = y$
$--$ **implies**
$--$ $y.f = c$

Thus in order to perform Hoare-style reasoning for a program, whose routines are specified with their preconditions and postconditions, we propose (at least) to compute aliases that may hold just before and just after each routine call. (Other details of Hoare-style reasoning with aliases are to be elaborated in future.)

III. Handling procedure calls in the non-object-oriented alias calculus

[2] introduces the following rule for a call $\mathbf{call}\ r(l)$ to a procedure r with a list l of actual arguments:

$$(a \gg \mathbf{call}\ r(l)) \ = \ (a[r^\bullet : l] \gg \underline{r}),$$

where

- $r^\bullet$ is the list of formal arguments of the procedure r,
- $\underline{r}$ is the body of the procedure r,
- $a[\tilde{l} : l] \ = \ (\ldots(a[\tilde{x}_1 : x_1])\ldots)[\tilde{x}_m : x_m]$ for lists of variables $\tilde{l} = (\tilde{x}_1, \ldots, \tilde{x}_m)$ and $l = (x_1, \ldots, x_m)$ (see the definition of $a[x : y]$ for variables x, y in Section I).

We elaborate on how to extend this rule to handle possibly mutually recursive procedures. First of all, from Section II we know that we must compute alias relations that may hold just before and just after each procedure call of a program. Next we introduce some definitions and notation and then propose an algorithm that computes required alias relations.

Any alias relation that may hold at the program point just before/after (the occurrence of) the procedure call c is called the *input*/*output* alias relation for (the occurrence of) the procedure call c. (In this definition and in what follows we assume that the aliasing semantics of (possibly mutually recursive) procedures is intuitively clear, however the semantics is to be precisely defined in our future work.)

Let $c_1, \ldots, c_n$ be all the occurrences of procedure calls in the program. Let c_0 be an (implicit) occurrence of the *Main* procedure call, which is performed when the program starts its execution.

For each $j = 0, \ldots, n$ we need to compute

(1) the maximal (w. r. t. inclusion) input alias relation $a^{c_j}_{max}$ for the procedure call c_j (obviously, $a^{c_0}_{max} = \emptyset$) and
(2) the maximal output alias relation $A^{c_j}_{max}$ for the procedure call c_j.

To obtain these alias relations, Algorithm 1 takes into account all possible sequences of procedure calls by iteratively computing an input alias relation a^{c_j} and output alias relation A^{c_j} for the procedure call c_j ($j = 0, \ldots, n$) so that at the termination of the algorithm

$$a^{c_j} = a^{c_j}_{max} \ \text{ and } \ A^{c_j} = A^{c_j}_{max} \ \text{ for each } j = 0, \ldots, n.$$

For an occurrence c of a procedure call in the program we denote by r^c the procedure and by l^c the list of actual arguments of the call.

Algorithm 1 Computes the maximal input and maximal output alias relation for each procedure call.

---

(1) for each $j = 0, \ldots, n$
  (1.1) $a^{c_j} := \emptyset$;
  (1.2) $A^{c_j} := \emptyset$;
(2) while all A^{c_j} ($j = 0, \ldots, n$) are not stabilized
  (2.1) for each $j = 0, \ldots, n$
    (2.1.1) $A^{c_j} := (a^{c_j}[(r^{c_j})^\bullet : l^{c_j}] \gg \underline{r^{c_j}})$,
      where $a^{c_j}[(r^{c_j})^\bullet : l^{c_j}] \gg \underline{r^{c_j}}$ is computed using calculus rules and whenever a subtask of computing $\bar{a}^{c_k} \gg \mathbf{call}\ r^{c_k}(l^{c_k})$ is encountered at a program point just before c_k (for some k):
      (2.1.1.1) $\bar{a}^{c_k} \gg \mathbf{call}\ r^{c_k}(l^{c_k})$ is treated as A^{c_k};
      (2.1.1.2) $a^{c_k} := a^{c_k} \ \cup \ \bar{a}^{c_k}$.

Algorithm 1 terminates since the set of expressions (i. e., variables in case of the non-object-oriented alias calculus) that

may be in the computed alias relations is finite. Note also that if all A^{c_j} $(j = 0, \ldots, n)$ are stabilized, then obviously all a^{c_j} $(j = 0, \ldots, n)$ are stabilized too. The algorithm is a variant of the Chaotic Iteration algorithm (see [10]) and it is subject to some optimization, e. g., maintaining a worklist of what really needs to be recomputed.

A. An example.

Let us illustrate how Algorithm 1 works on the model language program given on Fig. 1.

```
procedure Main   -- c_0
    then
        x := y
    else
        x := a;
        call q   -- c_1
    end
end
procedure q
    x := b
    then
        call Main   -- c_2
    else
        a := c
    end
end
```

Fig. 1. An example program [2, Example 13].

1. For $j = 0, 1, 2$: $a^{c_j} := \emptyset$; $A^{c_j} := \emptyset$.
2. $A^{c_0} := (a^{c_0} \gg \underline{r^{c_0}}) = (\emptyset \gg \underline{Main}) = (\overline{x, y},\ (\overline{x, a} \gg \textbf{call}\ q)) = \overline{x, y}$
   as currently $(\overline{x, a} \gg \textbf{call}\ q) = A^{c_1} = \emptyset$;
   $a^{c_1} := \overline{x, a}$.
3. $A^{c_1} := (a^{c_1} \gg \underline{r^{c_1}}) = (\overline{\overline{x, a}} \gg \underline{q}) = ((\overline{x, b} \gg \textbf{call}\ Main),\ \overline{x, b} \gg (a := c)) = \overline{x, y}, \overline{x, b}, \overline{a, c}$
   as currently $(\overline{x, b} \gg \textbf{call}\ Main) = A^{c_2} = \emptyset$;
   $a^{c_2} := \overline{x, b}$.
4. $A^{c_2} := (a^{c_2} \gg \underline{r^{c_2}}) = (\overline{\overline{x, b}} \gg \underline{Main}) = (\overline{x, y},\ (\overline{x, a} \gg \textbf{call}\ q)) = \overline{x, y}, \overline{\overline{x, b}}, \overline{a, c}$
   as currently $(\overline{x, a} \gg \textbf{call}\ q) = A^{c_1} = \overline{x, y}, \overline{x, b}, \overline{a, c}$;
   $a^{c_1} := \overline{x, a}$.
5. $A^{c_0} := (a^{c_0} \gg \underline{r^{c_0}}) = (\emptyset \gg \underline{Main}) = (\overline{x, y},\ (\overline{x, a} \gg \textbf{call}\ q)) = \overline{x, y}, \overline{\overline{x, b}}, \overline{a, c}$
   as currently $(\overline{x, a} \gg \textbf{call}\ q) = A^{c_1} = \overline{x, y}, \overline{x, b}, \overline{a, c}$;
   $a^{c_1} := \overline{x, a}$.
6. $A^{c_1} := (a^{c_1} \gg \underline{r^{c_1}}) = (\overline{\overline{x, a}} \gg \underline{q}) = ((\overline{x, b} \gg \textbf{call}\ Main),\ \overline{x, b} \gg (a := c)) = \overline{x, y}, \overline{x, b}, \overline{a, c}$
   as currently $(\overline{x, b} \gg \textbf{call}\ Main) = A^{c_2} = \overline{x, y}, \overline{x, b}, \overline{a, c}$;
   $a^{c_2} := \overline{x, b}$.
7. $A^{c_2} := (a^{c_2} \gg \underline{r^{c_2}}) = (\overline{\overline{x, b}} \gg \underline{Main}) = (\overline{x, y},\ (\overline{x, a} \gg \textbf{call}\ q)) = \overline{x, y}, \overline{\overline{x, b}}, \overline{a, c}$
   as currently $(\overline{x, a} \gg \textbf{call}\ q) = A^{c_1} = \overline{x, y}, \overline{x, b}, \overline{a, c}$;
   $a^{c_1} := \overline{x, a}$.

Items 8–10 (not shown) are the same as items 5–7 respectively, so $A^{c_0}, A^{c_1}, A^{c_2}$ given in items 5–7 are stabilized and equal to the maximal output alias relations sought for; the maximal input alias relations are $a^{c_0}, a^{c_1}, a^{c_2}$ given in items 1, 7, 6 respectively.

IV. Conclusion and future work

We are elaborating on the alias calculus and in this research-in-progress report presented how aliases might be used in Hoare-style reasoning, derived at what program points we were to compute aliases and proposed an algorithm that computed the required aliases.

Future work includes:

- elaborating on the overall process of Hoare-style reasoning using the alias calculus;
- precisely defining the aliasing semantics of (possibly mutually recursive) procedures; optimizing Algorithm 1, which computes alias relations for calls to (possibly mutually recursive) procedures; proving its correctness; and adopting the algorithm for qualified calls;
- coping with the infinity of some alias relations in the object-oriented alias calculus (e. g., after assigning $cur := first$ and then iterating $cur := cur.next$, the variable cur may be aliased to any element of the infinite set described by the regular expression $first(.next)^*$).

Acknowledgment

The author would like to thank his scientific supervisor Professor Bertrand Meyer, the head of the Software Engineering Laboratory at Saint Petersburg National Research University of Information Technologies, Mechanics and Optics, and Alexander Kogtenkov for helpful discussions. The Software Engineering Laboratory is supported by a grant from Mail.Ru Group.

The author is grateful to the anonymous referees for their useful comments on the original version of this paper.

References

[1] B. Meyer, "Towards a theory and calculus of aliasing," *Journal of Object Technology*, vol. 9, no. 2, pp. 37–74, 2010.

[2] ——, "Steps towards a theory and calculus of aliasing," *Int. J. Software and Informatics*, vol. 5, no. 1-2, pp. 77–115, 2011.

[3] ——, "Towards a calculus of object programs," *CoRR*, vol. abs/1107.1999, 2011.

[4] J. C. Reynolds, "Separation logic: A logic for shared mutable data structures," in *LICS*. IEEE Computer Society, 2002, pp. 55–74.

[5] S. Sagiv, T. W. Reps, and R. Wilhelm, "Parametric shape analysis via 3-valued logic," *ACM Trans. Program. Lang. Syst.*, vol. 24, no. 3, pp. 217–298, 2002.

[6] D. G. Clarke, J. M. Potter, and J. Noble, "Ownership types for flexible alias protection," *SIGPLAN Not.*, vol. 33, no. 10, pp. 48–64, Oct. 1998. [Online]. Available: http://doi.acm.org/10.1145/286942.286947

[7] I. T. Kassios, "Dynamic frames: Support for framing, dependencies and sharing without restrictions," in *FM*, ser. Lecture Notes in Computer Science, J. Misra, T. Nipkow, and E. Sekerinski, Eds., vol. 4085. Springer, 2006, pp. 268–283.

[8] Eiffel verification environment. [Online]. Available: http://eve.origo.ethz.ch

[9] C. A. R. Hoare, “An axiomatic basis for computer programming,” *Commun. ACM*, vol. 12, no. 10, pp. 576–580, 583, 1969.

[10] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of program analysis (2. corr. print)*. Springer, 2005.

Internal and online simplification in genetic programming: an experimental comparison

Yaroslav Borcheninov, Yuri Okulovsky

Ural Federal University
Yekaterinburg, Lenina str. 51
Email: yuri.okulovsky@gmail.com

***Abstract*—Genetic programming is an evolutionary algorithm, which allows performing symbolic regression — the important task of obtaining the analytical form of a model by the data, produced by the model. One of the known problems of genetic programming is expressions' bloating that results in ineffictevely long expressions. To prevent bloating, symbolic simplification of expression is used. We introduce a new approach to simplification in genetic programming, making it a uniform part of the evolutionary process. To do that, we develop a genetic programming on the basis of transofmation rules, similarly to computer algebra systems. We compare our approach with existed solution, and prove its adequacy and effectiviness.**

***Index Terms*—genetic programming, symbolic computations, computer algebra systems**

I. Introduction

Symbolic regression is an approach to data mining, which accepts a data, generated by some model, and produces an analytic form of this model. Probably, the most known and earliest successful application of the symbolic regression is Johannes Kepler's astronomical laws, which mathematically describe observations made by Tycho Brahe. Symbolic regression is an important step in the scientific method that prescribes explaining observed data through the construction of their mathematical model. By the close examination of such mathematical model, scientists understand its internal structure and suggest hypotheses about their underlying nature.

We should stress the difference between the symbolic regression and numerical regression methods, like the linear, segmented linear or polynomial regression. In case of numerical regression the model is fixed, and only its quotients are to be found. For example, by applying polynomial regression to the data, we explicitly suggest that the model is a polynomial function. If the actual model is a trigonometric function, line sinus, the regression can be made arbitrarily accurate by choosing the appropriate polynom's degree. However, no matter how accurate it is in the sense of mean square error, the polynomial regression is still incorrect, because it will unavoidable miss the fact that the observed model is the trigonometric function. Symbolic regression allows finding the model itself, and therefore the sinus function will be recognized as sinus.

Until recently, the symbolic regression could be performed only manually, and no algorithm of symbolic regression was available. With the discovery of genetic programming technique by John Koza [Poli et al., 2008], it becomes possible to automate symbolic regression. Now automated symbolic regression is widely used in natural sciences [Schmidt and Lipson, 2009], robotics [Robertson and Dumont, 2002], economics [Koza, 1994], medicine [Zhang and Wong, 2008], etc.

The algorithm processes versions about the actual data's model. These versions are expressions, encoded as trees and stored in the *pool*. Initially, these expressions are random. Then, the algorithm alters expressions with the following procedures.

- *Mutation*. The randomly chosen expression is changed by a replacement of a node.
- *Crossover*. Two randomly chosen expressions exchange subtrees.
- After all the mutations and crossovers are performed, the resulting expressions' set is subjected to the *selection*, which evaluates how each expression fits the experimental data. The least valuable expressions are then removed from the population.

With the time, expressions become better until the satisfiable solution is found.

The known problem of genetic programming is expressions' bloating, which means that expressions become ineffectively long. For example, expression $(x+1)^2-(x-1)^2-3x$ is bloated, because it actually equals to x and should be replaced by x in the pool. One result of bloating is unacceptable form of the algorithm output. It can be resolved with the simplification of the algorithm's result. However, bloating also hampers the algorithm's work by increasing the expression length and therefore the time required to compute them, and also by leading the algorithm along the blind alley. It can be resolved with the online simplification [Zhang et al., 2006], [Kinzett et al., 2008], when all expressions in the pool are simplified with some frequency. There exist other approaches ([Poli et al., 2008], [Mori et al., 2009]), however online simplification is considered to be more effective.

We argue that online simplification is too rough. Simplifying the expression inevitably leads to the elimination of potential growing points. For example, while approximating the function $(x+1)y^2$, the intermediate solution $(1+1)y^{1+1}$ can be found. This solution will be simplified to $2y^2$, which requires at least two mutations to become a correct answer,

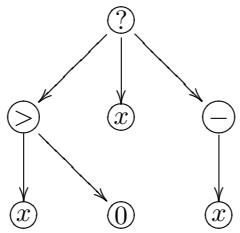


Fig. 1. The tree representation of the function $f(x) = |x|$.

e.g. $2y^2 \Rightarrow xy^2 \Rightarrow (x+1)y^2$. The initial solution $(1+1)y^{1+1}$ requires only one mutation $(1+1)y^{1+1} \Rightarrow (x+1)y^{1+1}$. Hence, the simplification hampers the evolution in this case. On other hand, the partial simplification $(1+1)y^{1+1} \Rightarrow (1+1)y^2$ does not produce such effect for the function $(x+1)y^2$, but does so for $2y^{x+1}$. Therefore, the question of where to apply the simplification depends on the problem specification, on the particular found expression, etc. In other words, the simplification can alter evolution of expressions in the same way the mutation and crossover do.

In [Borcheninov and Okulovsky, 2011], we introduce an approach of integration of simplification into genetic programming as uniform part. We call our approach internal simplification genetic programming (ISGP), as opposed to online simplification genetic programming (OSGP). The key aim of this paper is to measure the advantage of ISGP in comparison with OSGP.

Simplification is based on the rules, which describe ways of correct expressions' transformation. Since we use the simplification inside the algorithm, we must base our algorithm on the rules. In section 1, we show how to implement OSGP and ISGP an instances of more general rule-based algorithm. In section 2, we describe experiments to compare internal and online simplification.

II. Algorithm essentials

A. Expressions, trees and rules

An expression is represented as a tree of *nodes*. The example of such tree that encodes the function $f(x) = |x|$ is shown in the Fig. 1. Three types of nodes are considered: *constants*, *variables* and *operators*. In Fig. 1, node $\textcircled{x}$ is a variable node, $\textcircled{0}$ is a constant node. The remaining nodes are operators: addition $\textcircled{+}$, comparison $\textcircled{>}$ and ternary logical operator $\textcircled{?}$, defined as follows

$$?(x,y,z) = \begin{cases} y, & \text{if} \quad x \\ z, & \text{if} \quad \neg x \end{cases} .$$

Each node has a *return type*, which is an arbitrary C# type. Different return types can be used in one expression. For example, in Fig. 1, all nodes have `double` return type, except for the node $\textcircled{>}$ that has the `bool` return type.

We define numerous rules to transform these expressions. Some of these rules are universal, and can be applied to the tree regardless of data types or operations that are used in it.

(I-Re)	**select**	?A(?B)
	where	A.Type=B.Type
	mod	A→B
(I-Cr)	**select**	?A,?B
	where	A.Type=B.Type
	produce	A→B; **ret** A.Root

In I-Re rule, the **select** clause specifies the nodes that will be selected as a tuples (A, B), and then processed by the rule. The notion `?A(?B)` specifies that A is an arbitrary descendant of root (i.e., and arbitrary node in the tree), and B is an arbitrary descendant of A. Then, selected tuples are subjected to selection according to **where** clause. In I-Re, we accept only the tuples (A, B) such that they returning types coincides. To selected tuples, we can apply **mod** clause. In the case of I-Re, it replaces A with B. The tree remains correct, because of the selection in **where** clause. In I-Cr rule, the select clause `?A,?B` denotes that the rule accepts two trees, and selects an arbitrary node from each of them. Therefore, this rule is binary, while I-Re rule is unary. Then we demand the equality of their returning types, and finally replace A with B and return the root of A as an output. Using **produce** clause means that we specify directly the output of the rule. It is necessary, because binary rules accept two trees, and it is not clear which one of them should be the output.

Most of the rules, however, are not universal. With each data type T, the following rules are associated

(I-Co)	**select**	?A
	where	A.Type=T
	mod	A→new Const(v)
(I-Va)	**select**	?A
	where	A.Type=T
	mod	A→new Var(i)
(I-Tu)	**select**	?A
	where	A is Const
	mod	A→new Const(R(A.Value))

I-Co rule replaces the node with the return type T with the constant of the same type. Here v is a randomly selected value of the constant. I-Va rule replaces the node with the return type T with the variable. The argument i is a number of the variable in the argument array of the expression. Instances of I-Va rule have to be created for each variable of type T.We can also define tunning rules that adjust the constants. For Boolean and integer data types, such rules seem to be redundant, because they are just instances of I-Co. However, for floating point data type, rule I-Tu can be written. Here R is a random function $R(x)$ that returns a random number from $[x(1-c), x(1+c)]$. I-Tu rule allows changing the constant value gradually, near its initial value, and therefore differs from I-Co rule that does not take the previous value into account.

Some rules are even more specific, and are associated not with data types, but with the operations domain. The domain is a set of operations that are commonly used together and are bound by some mathematical laws. Examples are arithmetic

domain (addition, multiplication, etc.); trigonometric domain (sinus, cosinus, etc.); logical domain (conjunction, negation, etc.).

For each operation, we need an introduction rule. Two approaches to operation's introduction are possible.

	select	?A
(G-In)	**where**	A.Type = double
	mod	A→new Mult(A,new Const(1))

	select	?A
(G-In*)	**where**	A.Type = double
	mod	A→new Mult(A,new Const(v))

G-In rules selects a node with floating point return type, and replaces it with a new multiplication operation. G-In rule differs from all the rules above, because it does not change the function, encoded by the expression. It only inflates the expression and adds potential growing point in it. Of course, we could combine G-In rule with I-Co, therefore obtaining G-In*. However, it is not convenient. Suppose our task is to transform x into $2x$. With the modified G-In rule, we need the double luck to do that: we need to guess correctly both the operation and the constant. Wrong choice of constant may lead to significant decrease of the expression correctness, and therefore the expression will be removed, without a chance to adjust the constant. Original G-In rule does not affect correctness, and therefore modified rule can remain in the pool for a long time, so different mutations by I-Co rule can occur in the future and a right constant has more chances to be chosen.

For each operation, we also define simplification rules, for example transforming a multiplication of two constants into a constants with their multiplication, or transforming the multiplication of any node and zero into zero. We call such simplifying rules S-rules. They are known from computer algebra systems, so we will not study them deeply. Some rules are developed not for a single operations, but for several operations in the domain. The example is distributivity of addition and multiplication, which is G-rule for transformation $a \cdot (b + c) \to ac + cb$ and S-rule for reverse transformation.

Aside from simplification rules, we can also define a crossover rules for domain, with a very natural meaning:

	select	A,B
(I-CA)	**where**	A.Type=double and B.Type=double
	produce	new Div(new Plus(A,B),2)

The absence of quotation marks before A and B means that they are not descendants of the root, but the roots themselves. Crossover I-CA is reasonable: if two expressions fit the task, their halfsum may fit even better.

B. Implementation of genetic programming algorithms

To define a concrete algorithm in the genetic programming algorithms' family, we need to specify the operations, mentioned in the Intoduction: mulation, crossover and evaluation. We define mutation and crossover operations on basis of rules collection. The algorithm has two sets of rules: the set of unary rules for mutation, and the set of binary rules for crossover. In order to perform mutation, algorithm randomly selects expressions for mutation. Then, for each expression, we randomly select a rule, and perform it to obtain a mutated expression. Correspondingly, in order to cross two randomly selected expressions, the algorithm chooses a binary rule from the collection and performs it.

From the start of observations it becomes clear that different rules must have different probability to be applied. Each rule has multiple tags that describe the place of the rule in our classification. Then we assign to each tag its weight, and calculate the weight of the rule as the product of associated tags' weights. The greater the rule's weight is, the more the probability of rule's application is.

The most important tags are Inductive and Simplification tags. Inductive tags marks all the rules, which enlarge the expressions (G-rules from section 1.1), or changes the function the expression encodes (I-rules). Simplification rules make the expression shorter (S-rules). The ratio of Inductive and Simplification tags κ is the first important parameter of our algorithm.

The evaluation of the expression is performed by calculating several metrics and obtaining their weighted total. The fitness metric describe, how good the found expression g fits given data $(x_{1,j}, \ldots, x_{n,j}, y_j)$, and is calculated as

$$\mu_f(g) = \left(1 + \sum_{i=1}^{n} (g(x_{i,1}, \ldots, x_{i,m}) - y_i)^2\right)^{-1}.$$

Taking the reciprocal value is important, because it allows bounding the value of ρ, and provides correspondence between a higher value of ρ and a better expression. The length metric μ_l is a reciprocal to the count of operations in g. Valuation of an expression is determined as a weighted total $e(g) = w_f \mu_f(g) + w_l \mu_l(g)$. The ratio between the fitness metric and the length metric $\lambda = w_l / w_f$ is the second important parameter of our algorithm.

To perform online simplification, we modify the described algorithm. First, only I- and G-rules are allowed to be used in the algorithm. Second, the weight of length metric is set to zero, because algorithm does not have necessary means to decrease the expression's length. Finally, after each ξ iterations, we apply a simplification algorithm to each expression in the pool. Namely, we apply S-rules to expression until it is possible, and return the resulting expression in the pool. Online simplification algorithm has only one parameter ξ.

III. Experimental results

We conducted the following experiments to compare online and internal simplification in genetic programming. At first, we prepared test sets to run the algorithm on. Then, we found the optimal parameters of both algorithms to fetch best performances. Finally, we compared the performance of both algorithms.

In order to achieve a reasonable ratio between the representativeness of experiments and the time of computations, we followed the guidelines below. We limited the domain

of expressions by algebraic expressions that contain addition, subtraction, multiplication, division and power operations and integer constants. The reason is limiting the amount of parameters of algorithms. Two parameters are unavoidable: length/evaluation metrics ratio λ and inductive/calculation κ tags ratio. Introducing floating point constants demands us to use tunning rule (I-Tu). Our observations showed that intensity of this rule should be much greater than others', in order to find the appropriate values of constants. This adds one more parameter. Correspondingly, the introduction of trigonometric functions leads to various expressions like $\sin(\sin(\cos(\ldots)))$, and therefore these operations need to have their own tag with reduced value. Therefore, widening the domain requires increase of parameters. Since we needed to obtain the optimal parameters in order to compare approaches, we decided to limit the domain.

On the other hand, we made a high demand to the algorithm's outcome. The algorithm was provided with a very strict amount of data points: 10 for unary function and 100 for binary. The amount of iteration was limited by 10000, which takes about 15 minutes to compute. We also demanded the algorithm to find the exact function, used to generate the data, not its good approximation. The function may be presented as different expressions, however. It is a very strict requirement: sometimes the algorithm found the solution that was very close to data (root mean error is about 2-3%), and nevertheless, we neglected such solution and demanded the exact solution to be found. Summarizing, we can say that algorithm had to find an exact function with a limited data set in a short time. We believed that the complexity of this task compensates the domain narrowness.

To build the test set we made a rundown over different expressions, tested them with our algorithm and therefore obtained a knowledge about "complexity" of these expressions in terms of the algorithm. The considered parameters of expressions was the number of expression's arguments; the number of operations, used in the expression; the level of white noise, applied to data. At first, we builded a random tree with desired count of operations and tested, if the expression truly depends on all its arguments. Then, we formed test set as an array

$$\begin{array}{llll} x_{1,1}, & \ldots, & x_{n,1}, & y_1 \\ x_{1,2}, & \ldots, & x_{n,2}, & y_2 \\ & \ldots & & \\ x_{1,m}, & \ldots, & x_{n,m}, & y_m \end{array}$$

where

$$\{(x_{1,j}, \ldots, x_{n,j} \mid j \in 1, \ldots, m\} = K^n,$$

the set K is $\{0, 0.1, \ldots, 1\}$, $y_j = f(x_{1,j}, \ldots, x_{n,j}) \cdot (1 - p + 2p\alpha)$, p is white noise level and α is a uniform random number between 0 and 1. If f cannot be calculated for some j, we dropped the expression and searched again. On each data set, we run the algorithm several times and measure the average success rate. If the algorithm had accidentally found the form of expression containing least operation that planned, the data set was also considered invalid and was excluded from experimental result.

Variable count = 1

	p=0	p=0.01	p=0.02	p=0.05
c=2	96.67	95.38	90.87	95.62
c=3	29.47	30.63	33.93	25.68
c=4	11.67	0	0	0

Variable count = 2

	p=0	p=0.01	p=0.02	p=0.05
c=2	98.62	97.39	98.24	98.67
c=3	25.58	39.46	31.19	29.02
c=4	2.4	2.45	3.65	5.74

TABLE I
SUCCESS RATE, IN PERCENTS, FOR DIFFERENT VARIABLE COUNT, COUNT OF OPERATIONS c AND WHITE NOISE LEVEL p

Function	Success rate, %	Description
$(x^2)(x+4)$	80	Simple polinom
$\frac{y}{2}(x+2)$	80	A simple polynom with two variables
x^{x-y}	80	A simple non-rational function
$x^2 - (y+3)$	50	Intermediate polynom with two variables
$\frac{x}{2(x-3)}$	50	Intermediate rational function
$(3x)^{2y}$	50	Intermediate non-rational function
$2xy - y - 2$	20	Hard polynom
$\frac{x(x+4)}{x-5}$	20	Hard rational function
$\left(\frac{x}{4}\right)^{4x}$	20	Hard rational function

TABLE II
FUNCTIONS, SELECTED TO TEST SET

For each set of parameters, we run 50 successful data set, and each data set was processed by the algorithm 10 times. Obtained result are presented in Table III. The overall tendency is clear. The complexity is determined mostly by count of operations, then by the level of white noise. Additional variables seem to reduce the complexity, probably because of widening data set from 10 to 100 samples. We can also conclude that the algorithm is functional, even though initial parameters could be far from optimal.

We selected 9 expressions as test set for the OSGP and ISGP comparison. Selected expressions are listed in Table III. We did not selected expressions with 0% success rate, because it this case the difference between hard and impossible is not clear. For the same reason, we omited expressions with 100% success.

We ran ISGP algorithm with different length/fitness metrics ratio λ and calculation/induction tags ratio κ and obtained the resuls, presented in Table III. We see that the algorithm is in tote stable, and its success rate varies in range 60–70%. It is unlikely to find some local maxima outside the considered parameters' range. Parameters λ and κ by definition are greater that zero. When $\lambda = 0$ or $\kappa = 0$, the simplification is simply not performed, and expressions bloat rapidly, blocking the algorithm. When $\lambda > 1$ or $\kappa > 1$, the simplification is too strong: by out observation, no expressions of length more than 3 can be produced. Therefore we believe that the best success

κ	0.01	0.02	0.05	0.1	0.2	0.4
$\lambda = 0.01$	66.67	71.67	64.44	63.89	63.33	56.67
$\lambda = 0.02$	67.22	68.33	66.11	62.22	65	59.44
$\lambda = 0.05$	66.67	66.67	67.22	67.78	64.44	58.89
$\lambda = 0.1$	66.11	62.22	63.89	70	60	56.11
$\lambda = 0.2$	63.89	66.67	65.56	64.44	62.22	53.89
$\lambda = 0.4$	68.89	66.67	65.56	66.11	65.56	62.78
$\lambda = 0.8$	66.67	63.33	65.56	65	67.22	61.67

κ	0.02	0.04	0.06	0.08	0.1
$\lambda = 0.01$	68.33	73.89	68.89	67.22	66.11
$\lambda = 0.0333$	66.11	63.89	62.22	63.33	68.89
$\lambda = 0.0666$	66.67	63.89	61.67	62.22	64.44
$\lambda = 0.1$	73.33	65	63.33	70	63.33

TABLE III
SUCCESS RATES, IN PERCENTS, OF OSGP WITH VARIOUS VALUES OF THE PARAMETERS κ AND λ. THE LOWER TABLE GIVES A CLOSER LOOK TO THE AREA, WHERE LOCAL MAXIMA SEEM TO BE.

$\xi = 10$	$\xi = 20$	$\xi = 30$	$\xi = 40$	$\xi = 50$	$\xi = 60$
36.11	55	70.56	70	71.67	68.89

$\xi = 70$	$\xi = 80$	$\xi = 90$	$\xi = 100$	$\xi = 160$
68.89	65	70	68.33	65.56

TABLE IV
SUCCESS RATES, IN PERCENTS, OF ISGP WITH VARIOUS VALUES OF THE PARAMETER ξ.

rate of our algorithm is about 70% on our test set.

For OSGP, we need to determine the count of iterations between simplifications, ξ. The results of OSGP for different ξ are presented in Table III. Again, it is unlikely that the optimal value of ξ is greater than 160, because such rare simplification is hardly noticeable. On other hand, when the simplification is performed too often ($\xi < 5$), long expressions are almost never appear in the pool.

In table III we present the success rate of best algorithm's variants on test set. We can conclude, that the algorithms are very close in terms of performance. It is also obvious that accurate choise of parameters is important, and improves effectiveness significantly, at least for some functions.

IV. CONCLUSION

The research, presented in this article, proves the internal simplification genetic programming to be an operational technique that prevents bloating of expressions and provides effective symbolic regression. The only way to implement ISGP is to found genetic programming on the basis of expressions' transformation rules, as it was described in section 1.

The performance comparison states that ISGP is not worse than existed online simplification approach. ISGP also open a road for further research in the following areas. At first, we plan to explore the more presice devision of rules into groups, and finding the appropriate tags for such devision. This task can be considered even for the algebraic domain: for example, we could consider different tags for I- and G-rules. For the greater domains, this task is even more important, because additional tags emerge anyway.

Function	ISGP, $\kappa = 0.5$, $\lambda = 0.1$	ISGP, $\kappa = 0.04$, $\lambda = 0.01$	OSGP, $\xi = 50$
$(x^2)(x+4)$	80	100	100
$\frac{y}{2}(x+2)$	80	100	100
x^{x-y}	80	100	100
$(3x)^{2y}$	50	100	100
$x^2 - (y+3)$	50	95	100
$\frac{x}{2(x-3)}$	50	95	95
$\left(\frac{x}{4}\right)^{4x}$	20	25	25
$2xy - y - 2$	20	50	5
$\frac{x(x+4)}{x-5}$	20	0	20

TABLE V
SUCESS RATES, IN PERCENTS, OF ALGORITHMS ON TEST SET. THE SECOND COLUMN REPRESENTS THE INITIAL SUCCESS RATES, GENERATED WHEN BUILDING TEST SET. THE THIRS AND FOURTH COLUMNS ARE BEST RESULTS OF OSGP AND ISGP, CORRESPONDINGLY.

The more intriguing branch of research is adjusting the tag's weights during the algorithm's work. The tentativ observations show that such adjusting can sometimes drive the algorithm out of the local minimun by speeding up induction, or narrow the search around the best expression by increasing of the fitness metric weight.

We also plan to develop a distributed version of our OSGP implementation, and test it in real-world problems, mostly from robotics field.

ACKNOWLEDGMENTS.

The work is supported by the program of President of Russian Federation MK-844.2011.1.

REFERENCES

[Borcheninov and Okulovsky, 2011] Borcheninov, Y. V. and Okulovsky, Y. S. (2011). Genetic programming with embedded features of symbolic computations. In *KDIR 2011 — Proceedings of the International Conferenceon Knowledge Discovery and Information Retrieval*.

[Kinzett et al., 2008] Kinzett, D., Johnston, M., and Zhang, M. (2008). Numerical simplification for bloat control and analysis of building blocks in genetic programming. *Evolutionary Intelligence*, 4.

[Koza, 1994] Koza, J. R. (1994). Genetic programming for economic modeling. In *Intelligent Systems for Finance and Business*.

[Mori et al., 2009] Mori, N., McKay, B., Hoai, N. X., Essam, D., and Takeuchi, S. (2009). A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 13(14):237–238.

[Poli et al., 2008] Poli, R., Langdon, W. B., McPhee, N. F., and Koza, J. R. (2008). *A Field Guide to Genetic Programming*.

[Robertson and Dumont, 2002] Robertson, A. P. and Dumont, C. (2002). Design of robot calibration models using genetic programming. In Mayorga, R. V. and Rios, A. S.-D. L., editors, *Proceedings of the Third International Symposium on Rob. and Autom.*, volume 3, pages 449–454.

[Schmidt and Lipson, 2009] Schmidt, M. and Lipson, H. (2009). Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85.

[Zhang and Wong, 2008] Zhang, M. and Wong, P. (2008). Genetic programming for medical classification: a program simplification approach. *Genetic Programming and Evolvable Machines*, 9(2):229–255.

[Zhang et al., 2006] Zhang, M., Wong, P., and Qian, D. (2006). Online program simplification in genetic programming. *Simulated Evolution and Learning - SEAL*, pages 592–600.

Execution Analysis of ARPC Programs in the Environment of the Recursive Parallel Programming*

A. G. Sedov
Yaroslavl State University
Yaroslavl, Russia
E-mail: agsedov@gmail.com

***Abstract*—This article examines the analysis of an execution of a program written in the recursive parallel RPC language and the RPC extension for algebraic calculation facilities.**

An RPC program execution model named a Trace Graph has also been examined, as well as tools for its construction and rendering.

The Trace Graph formal definition is given in terms of Recursive Parallel Program schemes described in [1].

Next, the conception of the RPC language extension named ARPC is offered, main additional statements are described.

Furthermore, the author made a review of changes both in the execution model and the execution model construction algorithm which could be caused by RPC extension.

I. Introduction

The recursive parallel programming is a fairly easy and somewhat elementary method of distributed system development. This method was first introduced in the second half of the XX century. It spares a programmer from the designing of a distributed system architecture, needed, for example, when using the MPI. Instead of it, a programmer breaks the calculation task into some parts — recursive parallel procedures, which will be distributed among calculation facilities as necessary by Recursive Parallel environment.

To develop this method, the conception of the Recursive Parallel C language (shortly RPC) was offered[2]. The language fulfills the requirements stated below:

- it is a recursive parallel language designed for a virtual multiprocessor computing system with dynamic parallelization, hereinafter referred to as a recursive parallel machine (RPM).
- a parallel RPC program can be translated by a standard C compiler both into a parallel code for RPM and a sequential code for a common computer.
- it should be a tool for the program concurrency analysis (and program models). The language should also support specified RPM program performance analysis (and program models).

To implement the RPC language, the RPM Shell environment was created [3].

* This work is supported by the RFBR grant # 11-07-00549-a.

It should be noted that a programmer needs some tools to check the effectiveness of the calculation distribution among processors. These are construction and rendering tools for the program execution model and a simulation tool. A Trace Graph serves as an execution model for the RPC programs.

The Trace Graph is a directed graph formed out of the vertices associated with events occurred during the program execution and edges which are associated with a control flow relation. The Graph is used for attaining a bunch of goals related to analysis of the recursive parallel program behavior and its debugging: simulation, examining of potential concurrency, collection and rendering the statistical data. Trace Graph construction is a process of gathering information about the stucture of recursive parallel program execution and its performance during the run. Supporting the Trace Graph construction and its rendering is an essential part of this environment.

All functionality applied to work with the Trace Graph was implemented in the old version of RPM environment. It contained a library for graph construction representing the Graph in a binary format, a Trace Graph rendering module, and an RPM simulator. These solutions became outdated because they are not compatible with modern operation systems. However, the recursive parallel programming conception is still relevant, and the key element of the environment — a library for concurrent execution of the RPC programs — keeps on beeing upgraded.

This paper describes a new version of the Trace Graph construction tool and the Trace Graph renderer. The new Trace Graph constructor and renderer use the old Trace Graph format to make it compatible with the RPM Shell environment.

There are two main future directions: the Trace Graph structure revision and the RPC language extension. Their overview is given in Sections IV and V. A new Trace Graph formal definition is given in terms of Recursive Parallel Program schemes (RPPS) examined in [1]. Being defined in that way, the Trace Graph can be easily associated, vertex to vertex, with the RPPS. This will make all execution analysis applications much more visual.

The RPC language extension is aimed at realizing the ideas stated in [2], making the RPC more powerful. The new

language will be named ARPC (Algebraic RPC) because it is intended for simplifying the development of applications for algebraic calculations. The RPPS extraction should be a feature of ARPC-to-RPC compiler. The requirements for the RPC extension are stated in Section V.

II. RPC PROGRAM STRUCTURE

In this section we are going to give some basic information about the RPC. A recursive parallel program may be thought of as a set of mutually recursive procedures.

The computational process which is performed in a computational system supporting the recursive parallel programming, is a hierarchical concurrent process. The process consists of a number of system functions (such as functions for memory access) and activations of procedures. We will define an activation as a parallel procedure invocation with specified input parameters. An activation which makes a recursive call is located higher in the hierarchy than a called activation. Any component that corresponds to procedure activation can be a hierarchical concurrent process. There are several types of parallel process components:

- concurrently invoked activations of user recursive parallel procedures;
- sequentially invoked activations of user recursive parallel procedures;
- operators for the concurrent shared memory access;
- *Wait()* operator. Being called from parent activation, the operator makes it wait for the completion of all child activations.

The activation of a procedure will be concurrent if it is called by using a special parallel process call operator (*PCall(ProcedureName)*). After invoking the parallel procedure call, computations will proceed without stopping until the synchronization point is reached. The procedure can be also invoked in a sequential mode, so its execution will start eventually in the same process, from which it was invoked.

The exit from the child procedure to the parent one returns the control to synchronization point. Computations in the parent procedure and the child ones can be concurrently executed.

A hierarchical model of the concurrent computations describes a process interaction as follows: each activation can have control relations only with the parent and child activations. The activation can only be completed when all its child activations are completed. The parent activation can pass data to the child activation, when it is generated, and it gets data from the child activation, when it is completed.

Every activation executed at any moment has a unique number.

One of RPC characteristics is a block of parameters — a structure containing local data for their transmission from a calling procedure to a called one, and vice versa. The name of a local variable of a given type (the name of a parameter block) is declared in the calling procedure. The access to the elements of the parameter block in a child procedure can be done only by the command *P_(elem)*, where *elem* is the name of a structure element.

Example 1: The structure of a recursive parallel program (RPP). Let us consider a recursive parallel program for the explicit solution of the heat conduction equation using the finite difference method. Its alghorithm consists in repeated computation of a new heat distribution array using the previous array and the initial condition arrays (the main function). The NextLayer procedure either divides a given array part or gets an array from the shared memory and executes computations.

```
struct NLParam
{
   int begin,end,min;
   int branching;
   float tau,h;
}
parallel(NextLayer, NLParam)
{
 if (P_(end) - P_(begin) > P_(min))
 {//If the segmentation
  //limit is not reached.
   for(int i=0;i<P_(branching);i++)
   {
    struct NLParam pbl;
    //...Copying the parameter
    // block to pbl.
    //Calculation of the begin
    //and end of the vector.
    pbl.begin = P_(begin)
    + i*(P_(end)-P_(begin))
    /P_(branching);
    pbl.end = P_(begin)
    + (i+1)*(P_(end)-P_(begin))
    /P_(branching);
    PCall(NextLayer, &pbl);
   }
  Wait(); //Synchronization
 }
 else
 {
  //Loading the array segment
  //from the shared memory.
  //Calculations and writing the results
  //to a temporary array in the shared memory.
 }
}
int main(...)
{
 //...Heat distribution array creation.
 //Writing the array to the shared memory.
 //...Temporary array creation.
 for(int i=0;i<IterCount;i++)
 {
   PCall(NextLayer, &pbl);
   Wait();
```

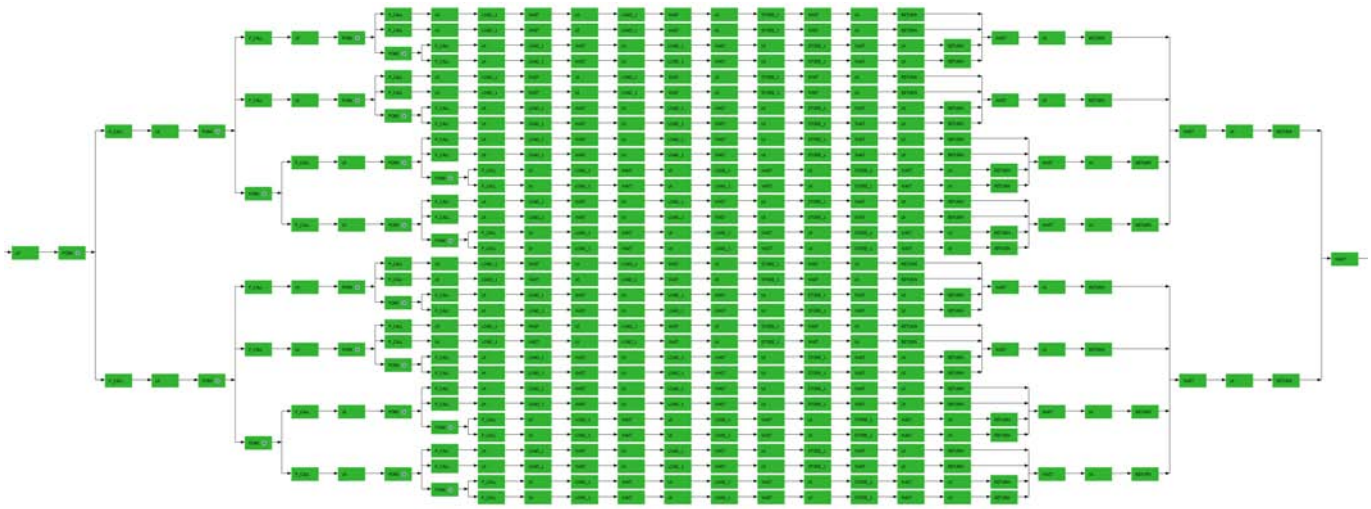


Figure 1. A fragment of the Trace Graph for the RPP from example 1 rendered in RPC Viewer (branching = 4).

```
   //Writing a new distribution
   //array to the file
   //Copying the temporary array
   //to the distribution array.
  }
}
```

III. Trace Graph construction and representation

The Trace Graph is a directed graph formed out of the vertices associated with the events occurred during the program execution, and the edges which are associated with transitions.

The following events are captured during RPC program execution, each event corresponding to a specific vertex type:

- forks — one or several calls to concurrent procedures; after call invocation, execution splits into a few parallel branches;
- synchronization operator calls — the parallel branches merge in a Wait vertex;
- calls to the shared and static memory management operations: allocation, deallocation, load, store;
- basic blocks — other sequential computations inside activations;

Specific data needed for modeling are saved as a file in addiction to vertices and relations between them. For example, a basic block vertex contains information about time taken to execute calculations. The vertices associated with the memory access operators contain statistics of the used memory volume.

A Trace Graph constructor is a library of functions for the sequential execution of an RPC program. It fully realizes the RPM functionality on a single computer. To compile an RPC program in a Trace Graph construction mode, the environment simply includes the library header file, instead of the header file for a standard mode library.

A Trace Graph is constructed by the GraphBuilder class. A GraphBuider instance is created before a call to the main program activation. As soon as any of the events from the list above occur, a new vertex is created and passed to the Graph builder.

The GraphBuilder handles an open-ended vertex list and a list of vertices that are ready to be written to a file. We will call a vertex open-ended, if new children may still appear. The open-ended vertex list contains one vertex from every process executed at the moment. We can consider it as a slice of the Trace Graph. After removing them from the open-ended vertex list, vertices are inserted into the ready-to-be-written list.

For each new vertex to be inserted into the open-ended list, its predecessor from the same activation is searched for. If a predecessor has not been found, the new vertex is recognized as a new activation beginning, and it is just inserted into the list. Else predecessor is removed from list.

To make a clear Trace Graph representation, the RPC-Viewer application was developed. It renders the Trace Graph as a hierarchical structure, allowing us to collapse or expand the chosen activations, and to view information mapped with the vertices. A format description for the Trace Graph file is passed to the RPC-Viewer in a special XML configuration file.

A Trace Graph rendering example can be seen in Figure 1.

IV. RPPS model

As we mentionted above, maintaining a Trace Graph construction in a format that is compatible with the old analysis tools was a goal of the first stage of work. However, the compatibility requirement appreciably limits the library facilities. For example, the information about associations between the Trace Graph structure and the code is not collected, so it is impossible to associate concrete recursive parallel procedures and their activations in the renderer.

Let us consider an RPPS (Recursive Parallel Program Scheme) model for describing the recursive parallel program structure and its execution in common terms.

A program scheme is a finite graph $G = (Q_G, q_0, \mapsto_G, L_G)$, where

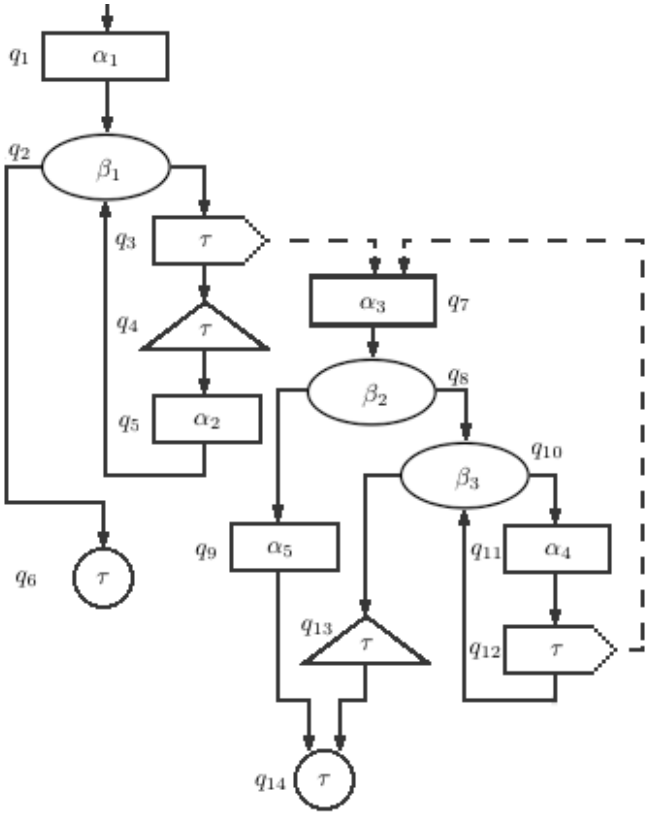


Figure 2. RPPS for example (1)

- $Q_G = \{q_0, q_1, ..., q_n\}$ — a finite set of vertices; every vertex being one of five types: action, selection, call, synchronization or ending;
- q_0 — the entry vertex (root);
- $\mapsto_G$: $Q_G \to Q_G^*$ — a function that maps every $q \in Q_G$ with its successors;
- $L_G : Q_G \to \Lambda_\tau \times Q_G^*$ — the function labelling graph vertices with symbols from Λ_τ. L_G also associates pcall vertices with the invoked vertex. Λ_τ is a basic set — alphabet containing abstract action names and silent actions τ.

A vertex q' is the successor of a q vertex, if $\mapsto_G$ assigns the q vertex to the vertex q'. The end vertices have no successors. Call vertices (pcall) are connected with two vertices: one vertex is the successor, another one is the called vertex.

The RPPS for the RPP of the heat equation solving can be seen in Figure 2. Here vertices q_1-q_6 correspond to the *main* function; q_7-q_{14} correspond to the *NextLayer* function. The solid lines designate $\mapsto_G$ transition, the dot lines designate L_G(Pcall).

A set of hierarchical states of $G \in RPPS_\Lambda\tau$ is a set $M_G = \{(q_1, \xi_1), ..., (q_n, \xi_n)\}$, where $q_1, ..., q_n$ are vertices from Q_G, and such that

- $\emptyset \in M_G$;
- $\xi_1, ..., \xi_n \in M_G$.

Example 2. The scheme from figure 2 generates $\xi = \{(q_4, \{(\tau, \{(q_9, \emptyset), (q_9, \emptyset), (q_9, \emptyset)\})\})\}$, a multiset corresponding to the situation in which the main function calls the Recursive Parallel procedure NextLayer, and NextLayer function, in turn, calls other three activations of NextLayer. These activations perform the calculations q_9.

Recursive-parallel program execution can be described as a sequence of hierarchical states $X = \xi_1 \longmapsto \xi_2 \longmapsto \xi_3 ... \xi_n$.

Let us give a formal definition of a new Trace Graph model.

We will define the Trace Graph of a recursive program as a finite graph $T = (Q_T, q_0, \mapsto_T)$, where

- $Q_T = \{q_0, q_1, ..., q_n\}$ — a finite set of vertices. There exists a function $f : Q_T \to Q_G$, such that it establishes a mapping of each vertex in Q_T to exactly one vertex in Q_G.
- q_0 — the entry vertex (root);
- $\delta : Q_T \to Q_T^*$ — a function which maps every $q \in Q_T$ to its succesors.
  If $f(q_i)$ is an action vertex, or a synchronization vertex, or a choice vertex, then $|\delta(q_i)| = 1$. If $f(q_i)$ is an end vertex, then $|\delta(q_i)| \leq 1$

The T structure differs from the existing Trace Graph structure first of all by a new vertex type, which was previously ignored during the Trace Graph construction — a choice vertex.

The RPPS for the RPC code will be constructed by means of the compiler. G_T will be shown in the renderer together with T.

V. A BRIEF OVERVIEW OF REQUIREMENTS TO THE RPC EXTENSION. ARPC.

To make the RPC language convenient for algebraic calculations, new statements are planned to be embedded. We will call the extended language Algebraic RPC (ARPC). Let us consider some of these statements.

A. Stencil statement

A stencil is a special statement that specifies the way in which the work is divided into parts in a procedure. The aim of introducing the stencils is to separate the logic of the recursive parallel dividing of work from calculations.

The ARPC is a macrodefinition language, where every stencil is a parameterized macro. Here, the notion of a macro has a more general meaning than it is usually accepted in C. The ARPC language allows us:

- to declare macros with parameters, where another macro can be used as a parameter;
- to use the nesting of macros;
- to declare a macro in a code both before and after its call;

For instance, we can write the *VectorDivision* stencil, which specifies the manner of vector division for the RPP from Example 1, giving us an opportunity to specify calculations on a lower level of the recursion later.

```
$stencil VectorDivision(procedure,begin,
            end,min,branching,Computing)
 if (P_(end) - P_(begin) > P_(min))
```

```
 {//If the segmentation
  //limit is not reached.
  for(int i=0;i<P_(branching);i++)
  {
   struct NLParam pbl;
   pbl.begin = P_(begin)+i*(P_(end)
   -P_(begin))/P_(branching);
   pbl.end = P_(begin)+(i+1)*
   (P_(end)-P_(begin))/P_(branching);
   PCall(procedure, &pbl);
  }
  Wait(); //Synchronisation.
 }
 else
 {
  $Computing
 }
}
```

Here, the *$stencil* macro declares a stencil named *VectorDivision* with input parameters *procedure, begin* e.t.c.

The *$def ... $endd* command is used in the ARPC to declare a macro, and the *$ins* command is used to include a library stencil.

```
struct NLParam
{
   int begin,end,min;
   int branching;
   float tau,h;
}
parallel(NextLayer, NLParam)
{
  $ins stencil VectorDivision(NextLayer,
  begin,end,min,branching,COMPUTING)
  $def COMPUTING//Macros definition
  //...calculations in the
  //deepest recursion level
  $endd
}
int main(...)
//...
```

The *$stencil* macro implementation requires solving a problem of limitations which will constraint the code to be substituted in the stencil and a problem of protection of variables the values of which should be affected only by the stencil logic.

B. Generic procedures

One more type of a parameterized macro which is planned to be embedded in the ARPC is a generic procedure. The purpose of this construction is to declare a recursive procedure, the structure of which depends on specific input parameters. In practice, it means that the compiler generates several procedures for different levels of the recursion from the macro.

The generic procedure can be used to vary the shared memory type, a task type and a semantic structure. To vary a procedure structure in accordance with parameters, the ARPC command *$if⟨condition⟩ ... [$else ...] $endif* should be used.

The way in which the compiler will reduce the number of generated procedures could be of great interest for further research.

C. Other suggestions

The RPC compiler implementation will probably allow to hide bulky work with parameter blocks. In [2] there are some other constructions, such as procedure specialization and a sticking-together command. They are planned to be developed further.

VI. Conclusion

In this paper we considered some requirements for RPC extension through macrodefinitions for algebraic calculations. For programs written in this language the automatic construction method of the execution model is described.

Let us take a look at the further research directions.

- The ARPC statement designing.
- The ARPC to RPC compiler implementation.
- The RPPS constructor implementation.
- Further research work with the Trace Graph format and renderer.
- Investigation of benefits of the created tools for the RPC program verification.

New opportunities and problems arise from expanding the RPC and implementation of the ARPC to RPC compiler. The compiler would enrich the Trace Graph construction algorithm by adding special trace operators to the RPC code. New ARPC statements should also be represented in the Trace Graph.

For the development of a verifying compiler we suppose to use of such formalisms as a hierarchical system of interacting automata or Nested Petri Nets[4].

References

[1] O. B. Kushnarenko *Recursive Parallel Program semantic and methods of it analysis* // Ph. D Thesis Grenoble, France 1997

[2] Badin N.M, Brodsky G.M., Sokolov V.A. *A Recursive Parallel Programming Language and its application to algebraic computations* // Joint NCC and IIS Bulletin Computer Science Vol. 11, Ershov institute of informatics systems SB RAS, Novosibirsk, Russia, 1999, pages 1-14.

[3] Vasilchikov V.V. *Parallel programming facilities for computing systems with dynamic load balancing* // Yaroslavl, YSU, 2001.

[4] I. A. Lomazova. *Nested Petri nets: modeling and analysis of the distributed systems with object-oriented structure*//Moscow, Science world, 2004

Towards a HLA-based Hardware-In-the-Loop simulation runtime

Eugene Chemeritskiy
The Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University
Moscow, Russia
tyz@lvk.cs.msu.su

This paper considers the possibility of the Distributed Real-time and Embedded (DRE) system simulation with CERTI, a general-purpose distributed simulation runtime based on the High Level Architecture (HLA) standard. Although it does not address DRE simulation and a number of issues should be resolved, the paper focuses on CERTI performance in the first place and compares it with a specialized DRE simulation tool. During the analysis of the comparison results, we make proposals on CERTI attuning and the design of an efficient HLA-based runtime.

Hardware-In-the-Loop Simulation; High Level Architecutre; CERTI;

I. INTRODUCTION

A. DRE simulation

Distributed Real-time and Embedded systems (DREs) are used in a wide range of electronic devices. Although DREs of household appliance usually contain only a few sensors and processing units, some complex equipment, e.g. onboard car, plane, and ship systems, is often composed of hundreds of devices connected to each other by dozens of diversified data transmitting channels.

The wide scope of DREs often imposes various restrictions on its components. Some DREs have to provide a certain performance, using a limited amount of power, resist an aggressive environment conditions, or keep within certain limitations on physical sizes and weight. In case the existing equipment does not meet these requirements, some devices have to be redesigned.

The missing DRE devices are usually developed by several independent workgroups. Their tasks have significantly varying complexity and assigned periods of time. As a result, devise prototypes constantly have different readiness degrees, and their joint trials are often impossible. However, the majority of errors are detected at the stage of component integration, when the prototypes are nearing its completion and the cost of error correction is reaching its maximum.

DRE simulation provides a common approach for early detection of integration errors. It includes the following steps. At the stage of DRE blueprinting, developers create its coarse component-wise simulation model. It serves for verification of the simplest DRE properties and detection of the related design errors. The further development goes, the further device models are refined. Simulation accuracy gradually grows, and verification reveals more and more tricky errors. Finally, as the hardware prototypes become available, they join the simulation instead of their software counterparts. To sum it up, exploring the current model properties at each stage, DRE developers can identify and shortly fix device integration errors as soon as it possible.

Considered Hardware-In-the-Loop Simulation (HILS) composes the software models with the hardware equipment and requires a specialized simulation hardware-software environment to interconnect them to each other. The software part of this environment, the simulation runtime encapsulates the details of communication with the diverse DRE components and provides a common API over it. However, the runtime cannot hide their difference completely. Hardware devices are often unable to work properly without a strict binding to astronomical clock. For example, its implementation may accept the reply for only a short period of time after the request send. These real-time constraints impose additional requirements to the simulation runtime. Therefore, HILS runtime significantly differs from a general-purpose one [1].

B. The HILS STAND simulation environment

For years the Computer Systems Laboratory (CS Lab) of Lomonosov Moscow State University has been developing its own hardware-software environment for Hardware-In-the-Loop simulation called HILS STAND [1]. Actually the roots of HILS STAND go to more general discrete-event simulation system DYANA [2], developed by CS Lab in early 1990s. Since, DYANA has been serving as a basis for a number of experiments on applicability of new approaches to simulation of the diverse computer systems. Started as one of the proof-of-concept DYANA branches, HILS STAND has been used to accomplish a number of HILS projects and gradually evaluated into a powerful simulation suite.

The core of the HILS STAND suite is formed by the specialized discrete-event runtime. This runtime provides several independent model execution modes. While in as-fast-as-possible mode, the HILS STAND runtime provides an efficient execution of software-only models and acts similar to a general-purpose runtime. The soft, hard and scaled real-time modes address the different kinds HILS in the first place. Being in any of this mode, the runtime executes device models

This work was supported in part by the Ministry of education and science of the Russian Federation under Grant “Development of an integrated environment and complex analysis methods for distributed real-time computer systems functioning”.

with the corresponding time constraints and allows their interactions with the connected hardware devices.

Besides the runtime, the suite includes a number of subsidiary simulation tools. For example, HILS STAND provides an integrated simulation data collector and a dynamic visualizer with the ability to modify model parameters during its execution and enables a human-controlled simulation. Although the runtime provides C++ API and encapsulates the details of interactions among the simulation participants, HILS STAND does not imply the model to be developed in a pure general-purpose language. Instead, it provides a specialized high-level model description language, its translator, and the corresponding IDE. Actually, these developments have a rich history either [3].

Because of the large number of DRE devices and a certain complexity level of their models, HILS STAND instances usually include a number of nodes connected by a private LAN. The nodes have a similar configuration, except some additional interfaces for external hardware simulation participants. The additional hardware channels can also interconnect several nodes directly. This configuration is used for experiments with the corresponding channel controllers and the other switching equipment [1].

C. An advanced simulation runtime

Although the HILS STAND is actively used in a number of different HILS projects, CS Lab has never ceased to follow the trends and test new simulation approaches. Recently it started new project on the rethinking of HILS environment and the development a new tool-chain, combining the experience gained in over twenty years of simulation experiments with latest applicable technology efforts. The research list of this large-scale project includes a new language for DRE simulation models; integrated trace tracking and visualizing tools; new verification engine; and an advanced HILS runtime, which is the subject of this paper.

During the last 60 years, discrete-event simulation runtimes made a significant advance [4]. There are several different classifications, but it is a common practice to divide them into a number of generations associated with some historical trends. One of the most noticeable trends now is a standardization of the runtime API. Thus, we believe it is giving a birth to the new runtime generation. Unfortunately, there is no any off-the-rack and well-fitted standard for HILS runtimes. However, exploring of adjoining simulation areas revealed some attempts to use High Level Architecture (HLA) for real-time simulation [5] and it is pretty close to HIL. So the idea of HLA adoption for HILS runtime appeared.

Although there are a lot of different HLA-based simulation runtimes, each of them requires a large amount of additional work. As it was shown by the analysis [6], CERTI happened to be the best initial approximation for the advanced HILS runtime we want to make. However, CERTI do not initially target real-time and HILS, and a number of related issues have to be resolved. Briefly considering their whole scope, the paper gives the first priority to performance of a HILS runtime. In particular, the paper introduces a couple of benchmarks that revealed significant advantage of HILS STAND over the CERTI. Due to a lower performance, out-of-the-box CERTI version cannot be used for the scope of simulation tasks HILS STAND can easily manage. Therefore, the paper contains the analysis of its architectural drawbacks and introduces a number of proposals on their reduction.

II. A HLA-BASED HILS RUNTIME

A. Standardization trends

Simulation as a method for exploration of diverse object properties and regularities among them outruns the advent of computers for many years. However, its rapid development started after the complex mathematical calculations had been assigned to fast and reliable computers. In the beginning of the 1950s, the term simulation acquired the default meaning of digital computer simulation. Subsequently the simulation was defined as a combination of designing of the observed system model and holding the necessary experiment set on digital computers [7].

From the very beginning of the simulation history the observed systems always tended to be represented in deeper detail level. This tension results in the increasing size and complexity of developed simulation model. This growth required a respective performance increase from computer systems, and this fact resulted in emergence of parallel simulation systems. These systems share the simulation task across multiple computing nodes. Typically such systems were implemented locally within the organization that wanted to use it.

The complexity of the models was not the only factor leading to computer simulation tool evolution. The scope of simulation has been growing either. After new simulation problem types appeared, the related requirements were imposed to modeling and simulation tools. For instance, distributed simulation is often required in case of joint product development when different product component are produced by a number of workgroups located in different organizations. This type of simulation intends encompassing of several geographically separated simulation systems, which in turn may consist of a single compute node, or be a parallel system. Historically, the appearance of this task type led to the creation of distributed simulation systems that provide an essential set of services to the simulation participants and ensure its consistent behavior [8].

Currently we believe that the next step in the runtime evolution is a standardizing of the distributed system interfaces. Uniform interfaces provide possibility to combine among a variety of independent simulation systems and create general models that can be handled by every distributed system corresponding to the standard specifications. One of these standards is described in the next section.

B. The HLA distributed simulation standard

HLA is a conventional standard in the field of distributed simulation. The roots for the HLA stem from distributed virtual environments. Such environments are used to connect a number of geographically distant users. The HLA standard is a conceptual heir of Distributed Interactive Simulation (DIS),

which is a highly specialized simulation standard in the domain of training environments [8]. The primary mission of DIS is to enable interoperability among separated simulation systems and to allow the joint simulation of their participation. HLA standard remains relevant to the DIS principles and even extends them.

HLA appeared in 1993, when the Defense Advanced Research Projects Agency (DARPA) designated an award for developing of an architecture that could combine all known types of simulation systems into a single *federation*. The HLA standard initially addressed all kinds of as-fast-as-possible, soft and hard real-time, discrete-event and time-driven, fully-synthetic, human- and hardware-in-the-loop distributed simulations. However, hard real-time constraints were not supported until the latest HLA standard version, namely IEEE 1516-2010 (Evolved) released in the very end of 2010 [9]. The majority of HLA-based simulation tools were built on the previous HLA standard versions and do not offer a full HLA Evolved support yet.

Thereby, HLA-based HILS became possible quite recently and any researches in this area are innovations in some sense. However, these researches seem to be prospective because of a number of benefits HLA gives. At first, HLA strict support by both the runtime and the models provides their guaranteed compatibility. It means that HLA model developed with one runtime can also be used with other runtimes without any modification. In fact, HLA forms an independent market of out-of-the-box simulation models which can be used with any HLA-compatible simulation runtime.

Secondly, HLA is used as an external simulation interface by some non-distributed runtimes. This peculiarity enables joined simulation encompassing diversified runtimes and, consequentially, different model types. For example, a single simulation can include both time-driven fully-synthetic and discrete-event hardware-in-the-loop models simultaneously, and their developers do not have to adjust their models for this cooperation.

In addition, there are a lot of subsidiary runtime-independent HLA-based simulation tools, such as statistic collectors, simulation analyzers, high-level model describing languages and corresponding IDEs. These tools operate at the model level over the HLA API and do not require any additional support from the simulation runtime. Therefore, they can be reused with any runtime implementation.

HLA specification introduces its own terminology generally used by the developers of HLA-based simulation tools, and CERTI is not an exception. Therefore, we include a short notion of fundamental HLA terms. The simulation runtime specified by HLA is named the Run Time Infrastructure (RTI). RTI provides services a number of joined federates - simulation participants of any kind. The association of all federates forms federation.

C. CERTI brief description

CERTI is a HLA-compliant RTI developed by the French Aerospace Laboratory (ONERA). The project started in 1996 and its primary research objective was the distributed simulation itself whereas the appeared HLA standard was the project experiment field. CERTI implementation started with the implementation of the small subset of RTI services, and was used to solve the concrete applications of distributed simulation theory [10].

Since the CERTI project was open sourced in 2002, a large distributed simulation developer community has been formed around the project. In many ways due to contributions of enthusiasts, the CERTI project has grown from basic RTI into a toolset including a number of additional software components that may be useful to potential HLA users.

The CERTI project has always served a base for researches in the domain of distributed simulation, and a number of innovative ideas have been implemented with its use. Thus, the problem of confidential data leak was solved in context of CERTI RTI architecture, and the considered RTI guarantees secure interoperation of simulations belonging to various mutually suspicious organizations [11]. The certain interest for the considered project is a couple of application devoted to high performance and hard real-time simulation.

In spite of HLA is initially designed to support fully distributed simulation applications, it provides a framework for composing not necessarily distributed simulations. Thereby there was created an optimized version of CERTI devoted to simulation deployed on the same shared memory platform and composed simulation running on high-performance clusters [12].

Some experience could also be adopted from ONERA project on simulation of satellite spatial system. Each federate in this federation is a time-stepped driven one. It imposes an additional requirement of hard real-time: the simulation system should meet the deadlines of each step and synchronize the different steps of the different federates [13].

Despite the distribution of commercial products, the project development is still continuing in accordance with the HLA simulation standard progress. Thus, CERTI supports HLA IEEE 1516-2000 version since 2010 in addition to previous DMSO 1.3 version.

D. Designing a CERTI-based Runtime

There are a lot of difficulties on a way to a CERTI-based HILS runtime. First of all, the supported version of HLA standard does not currently address real-time simulation and a fortiori it does not address HIL. First, IEEE 1516-2000 specifications do not provide any method to specify end to end prediction requirement for federate. Second, CERTI encodes reliable and best-effort transportation types with TCP and UDP network protocols which are not suitable for real-time simulation. Finally, CERTI works over the operating system and is unable to control its resources. All the listed paragraphs have a significant affection to amount and predictability of the runtime overhead crucial for any real-time simulation [5].

Second group of issues concerns the hardware integration during the HILS. The runtime should have an extendable support of the diverse data transmitting channels. This fact implies a number of additional restrictions to both hardware and software components of the simulation system. For

example, the hardware devices usually have strict data message format specifications. Therefore, RTI cannot use the only message to transmit both internal service data and federation one.

The final design challenge is the reuse of legacy tools from the HILS STAND software suite. Some of its components, such as the dynamic simulation visualizer and the generator of fault injections, cannot be efficiently implemented over the existing HLA interface and should be integrated into the RTI. Actually, their integration leads to additional research and development subtasks and requires a number of problems to be resolved.

Although each of the listed problems is important, this paper is devoted to the provision of the HILS STAND-comparable runtime performance level. Currently, HILS STAND is used to perform HILSs by a number of different DRE development projects and we are curios if the HLA-based simulation runtime is able to execute models with the similar complexities and real-time constraint sizes. The remainder of this paper is devoted to this issue.

III. CERTI PERFORMANCE EVALUATION AND ARCHITECTURAL ANALYSIS

A. Runtime benchmarking

During the HILS, each of the hardware participants interacts to the other DRE components according to a predefined time-related behavior specification. If the runtime does not meet requirements of this specification, the device may work incorrectly. It is simple to slow down a fast software model to correspond the device speed, but it is not possible to meet these requirements if the model works slower than the hardware expects. Thus, the speed of event handling is a crucially important property for any HILS runtime. Actually, its value can be used to determine the complexity of simulation tasks the runtime can efficiently solve. The smaller event handling time of the runtime, the wider range of simulation tasks it can solve. Moreover, the faster runtime works, the smaller its requirement to the hardware. For example, a slower runtime may need more nodes to run the same simulation model.

Making an assessment of CERTI applicability to the range of usual HILS STAND tasks, we choose two simple time-regulated client-server models from the HILS STAND test-suit and run them in as-soon-as-possible mode in both runtimes. Each of these models consists of a single server and a single client. The client sends messages to the server. Each message contains one integer parameter, whose value is decremented after each send, until it reaches zero. Thereby, the initial value of this parameter also sets a number of client messages to be transmitted. In the first model the server records the received values and works as a simple registrar. In the second model server also sends back to the client every message it receives, and the client do not send the next message until it gets a reply. The remainder of this paper refers these models as “Avalanche” and “Ping-Pong” tests respectively.

Although the described models are pretty simple, the similar simulation models are often used for the same purposes [14-15]. Federates of the Ping-Pong test are actually executed consequently. After the message send, client waits for a server reply. In a similar manner, server waits for the message instantly replies back to the client. Thereby, the time of simulation reflects the speed of message transmission rather accurate and can be used as a performance index for a runtime response time. Avalanche, in contrary, allows a fully parallel and logically unrestrained federate execution. Thus, the whole runtime can be considered as a media for data message transmission. Therefore, the simulation time can be treated as a reflection of a runtime throughput.

Both systems were tested using a hardware bench composed of two identical nodes. Each node ran a single model component, either client or server. The simulation time was measured by each model component independently of each other. The timer started right after the initial synchronization and stopped when the component had been ready to resign. Final results were formed as an average of two component readings for each model configuration.

As it is clearly shown by the benchmark results (Table I), overall CERTI performance is a several times lower than the one of HILS STAND. Although these results reduce the range of acceptable simulation tasks dramatically, the usual real-time requirements still accept CERTI as a HIL runtime. However, increase of its performance becomes an important direction of further development. The remainder of this segment presents a deeper CERTI analysis and introduces some proposals on its refinement.

TABLE I. THE AFFECTION OF MESSAGE NUMBER TO SIMULATION EXECUTION TIME, MS

Message number	**Avelanche**		**Ping-Pong**	
	CERTI	***HILS STAND***	***CERTI***	***HILS STAND***
10	4,1	1,6	10,2	2,3
100	38,1	7,6	94,4	22,8
1000	399,7	84,8	884,6	228
10000	6063	1127,6	8770,7	2280
100000	60601	11722,1	87643,2	22800

B. CERTI architecture analysis

Being a distributed simulation middleware, RTI provides a number of joined federates with API specified by the HLA standard. The main purpose of this API is to encapsulate any details of communication among the joined federates, network communication included. Thus, RTI includes a number of remote components corresponding to a number of federates joined. These components are generally known as Local RTI Components (LRCs).

Maintaining the federation consistency, RTI constantly synchs a set of joined federates. Therefore, the efficiency of their coordination affects the overall RTI performance significantly. Fully distributed architecture implies equal and self-sufficient LRCs, and its implementation requires complicated consensus algorithms. Developers usually avoid the excessive complexity by introducing the Central RTI Component (CRC) that stores shared data and implements some synchronization algorithms locally. Both centralized and decentralized RTI architectures have certain weak and strong sides, and their reasonable combination is a first cornerstone of

the efficient RTI implementation [13]. This segment considers the approach implemented by CERTI.

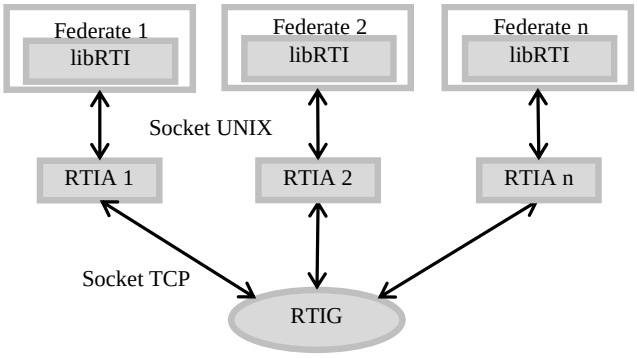


Figure 1. CERTI RTI architecture

Generally speaking, CERTI RTI consists of three components: RTI Gate (RTIG), RTI Ambassador (RTIA) and libRTI. RTIG is a process that runs on a separate host and serves as CERTI CRC. RTIA is a process that runs on the same host federate runs. Therefore, the number of RTIAs equals to the amount of joined federates. Both RTIG and RTIA are single-thread processes. Whereas they form RTI internals, libRTI runtime library implements API specified by HLA. libRTI links to the joined federate and connects it to the corresponding RTIA process by pipe (Unix socket). An aggregate of RTIA and libRTI forms CERTI LRC. Communication between CRC and LRCs goes through the network sockets [10]. Figure 1 presents a visual representation of the described architecture.

RTIA processes never communicate to each other directly. All data exchange among them goes through the RTIG. Thus, CERTI bases on a fully centralized architecture and its CRC component coordinates the joined federates single-handedly. Indeed, RTIG implements the most of RTI services whereas the sole purpose of RTIA and libRTI is the formation of a convenient communication infrastructure between RTIG and a number of joined federates. In other words, LRCs of CERTI generally serve as connectors between CRC and the end simulation participants.

Summing it up, CERTI RTI uses a fully centralized architecture and has a strong CRC. The simplicity of this organization gives a number of benefits to the RTI developers. First of all, concentration of all the control inside of a single process simplifies the implementation of RTI services. The absence of any direct links among the LRCs does not require network consensus algorithms and reduces the corresponding synchronization overhead significantly. Modification of a federation state made by RTIG is instantly propagated among all the related federates and all data received by LRCs can always be used without any additional conformation.

The second centralization profit is the simplicity and deterministic of communication among the distributed RTI components. Each data exchange among the joined federates always go through RTIG, where all the synchronization issues are solved locally and, therefore, do not require any network communication at all. Thus, coordination of the joined federates with RTIG always requires a certain and relatively small number of network messages, and uses network bandwidth rather efficiently. In contrast, fully distributed architecture, that include a number of equal self-sufficient LRCs and does not include any CRC, requires a complex synchronization algorithms, that are usually imply an intensive network communication to the consensus.

Third, strong CERTI centralization allows accelerated implementation and testing of the innovations. The proof-of-concept for a new runtime algorithm can be implemented locally inside RTIG. The developers get rid of complex network interactions and asynchronous changes of federation state. Their implementation should run in context of a single threaded process only. This peculiarity of CERTI, composed with its open source code, has proved this RTI as good foundation for the diverse simulation researches.

Finally, the centralized single-host execution of RTIG provides a convenient way to isolate the certain RTI subsystem and develop a number of diverse “surgical” benchmarks for them. The distributed kind of some RTI services (such as time and object management services) makes it really hard to get a fair estimation of their in-field implementation efficiency. The true results are often shaded and distorted by the network communication overhead that is hard to predict and, a fortiori, to avoid.

However, the same centralized architecture causes a number of problems. The most crucial of them is an excessive load of the RTIG. In case of small federations, composed of a few federates, RTIG does its service well. But the more federates join, the more RTIG is loaded, and the dependency is not linear. Each joined federates results in a new communication flow, and an increase in the complexity of its processing. For example, a single interaction send often leads to a number of notifications to its recipients. There comes a moment RTIG is unable to process the incoming data flow fast enough and becomes a bottleneck. In this case, it constantly makes the joined federates wait, and, as a result, federation is executed merely consequently. The next segment presents some approaches that can conceptually improve the current CERTI architecture and in some ways mitigate its problems.

IV. CERTI ATTUNING PROPOSALS

A. *Layered architecture*

As it was shown in the previous section, fully centralized architecture of CERTI results in an excessive CRC load during the execution of large federations, and it is a serious design drawback. There are a lot of RTI implementations that compose centralized and decentralized approaches in a more equitable way and get better performance in return [14]. However, architectural mixture requires a revolutionary alteration of CERTI internals and eliminates all the benefits of its clear and simple component structure. This section presents an alternative approach to CRC load reduction based on a layered RTI architecture.

Although each federate represents a certain component of real system, the level of their abstraction can be volatile and do not reflect logical structure of the system. For example, a

model of onboard system may include one coarse federate corresponding to a number of its secondary subsystems and a bunch of fine-grained federates corresponding to components of the most important subsystem simultaneously. Fine-grained federates are clearly less abstract than the coarse one because they correspond to smaller elements of the logical structure. Only their aggregate may form the new logical subsystem that can interact to the other subsystems on equal footing. Thus, federates of the aggregate depend on each other. They are logically linked.

During the simulation federate aggregation can be distinguished by the intensity of their interaction. As it is shown by the practice, members of a certain aggregate interact to the each other far more frequently than to any external federate. Federates are clustered into a number of aggregates and encapsulate the majority of communication traffic inside of them. Only a little part of traffic goes beyond and connects members of different aggregates.

CERTI has a centralized architecture and implements federate interaction of any kind using its CRC regardless to the model logical structure. In case the number of data exchanges is large enough, CRC is overloaded, becomes a bottleneck, and slows the simulation down. However, federates do not really care about the inner communication of aggregates they do not belong. Thus, the simulation traffic can be separated according to the model logical structure. The only thing we need is a dedicated middle-level CRC for each federate aggregation. From one hand, it will control the aggregated federates, encapsulate their inner traffic, and take some load of the real CRC. From the other hand, the real CRC will see it as a regular federate that runs in accordance to the common HLA execution rules and generates traffic flow corresponding to the bunch of aggregated federates. The middle-level CRC can be implemented as a new LRC frontend and does not result into significant increase of the RTI complexity.

There are several natural extensions of the described idea. First, federates can be clustered by a number of attributes differed from logical structure of the simulated system. For example, the non-uniform distribution of the federate communication intensity is a sufficient aggregation criterion. Second, the same trick can be used several times. In their turn, aggregated federates can be separated into a number of smaller groups, and form a new simulation control layer. Therefore, the described RTI architecture is referenced as a layered architecture.

To sum it up, introduction of the layered architecture results into a number of benefits. First of all, it solves the excessive CRC load problem and increases scalability of RTI. Indeed, the middle-level CRCs process the internal aggregate traffic independently and take a part of responsibilities from the real CRC. Each middle-level CRC can be executed by a separate host, thus, the RTI control is distributed automatically without any data replication or sophisticated coherence control algorithms.

Second, aggregation allows reducing of the synchronization losses. Internal interaction of aggregated federates goes through the middle-level CRC and does not take into account the most of external dependencies. Therefore, it is more efficient than the regular one. However, their external interaction is less efficient and includes two mediators, namely, the middle-level CRC and the regular one. If aggregation can be chosen *sufficiently* well, the synchronization losses can be reduced respectively.

Third, aggregation allows accurate RTI attuning. Federates can be clustered according to a set of services they use. Unused RTI components can be safely removed from the middle-level CRC and its complexity will reduce respectively. The remaining services can be also attuned to the requirements of the joined federates. For example, each middle-level CRC may implement its own time management algorithm that is effective for the aggregated federates independently of other RTI components. In case of HILS, the main CRC should always use conservative time management algorithms as a core. However, middle-level CRC component may use optimistic algorithms in case it is more efficient.

Finally, aggregation is a way to increase the efficiency of interactions within a single node. Centralized architecture does not take into account the relative position of federates. Even in case they are running on a single node, every data exchange goes though the RTIG. Thereby, RTI uses two network communications to transmit data between two processes on a single host. This wasteful data handling results into a significant performance decrease. Aggregation of all federates on the node actually allows their direct interaction without any network involvement. Thereby, the concept of node aggregation brings some advantages of the decentralized peer-to-peer architecture without any changes in a current CERTI logic.

The weak side of the layered RT architecture is indeterminism of RTI structure and its dependence on the executed model structure. It also requires some automated static and dynamic model analysis tools responsible for the criterion selection and the corresponding suboptimal federate segregation. However, the benefits it may give seem to worth efforts, and this approach appears to be rather prospective.

B. Thread-based LRC

CERTI LRC consists of libRTI library and RTIA process connected by UNIX pipe. Although libRTI is linked to federate process and provides HLA API, the library does not really implement RTI logic. The library just redirects the incoming method calls to the connected RTIA. In more details, every time federate calls RTI service, libRTI sends to RTIA a message with the associated method identifier and a set of supplied arguments. RTIA handles these queries and replies back with results. Thereby, libRTI can be considered as LRC frontend whereas RTIA corresponds to LRC backend. Due to this modular LRC structure, changes of HLA API will not affect the LRC backend directly and the corresponding RTI changes will require a minimum of effort. Currently CERTI uses this flexibility to maintain both DMSO 1.3 and IEEE 1516 2000 HLA versions.

Another advantage of the two-component LRC against the single-component one is the increase of simulation security and reliability. Both libRTI and RTIA run in their own context and verify every incoming message. Therefore, there is no way the

federate can read or modify any internal RTI data, except the calls of the HLA API. For the same reasons, failure of the joined federate never leads to a failure of the whole RTI.

However, the flexibility of the composed LRC decreases an overall RTI performance. Every time the federate calls RTI method, at least two internal messages are generated, and this number may increase in case of RTI callback requests. Transmission through the pipe requires message parameters to be serialized during the send and deserialized on its reception. These data format conversions inevitably result into undesirable memory copying and additional CPU load.

There are several ways to avoid the unnecessary communication overhead. The first one is to replace the pipe with a set of queues in a shared-memory. This approach does not require any data reformatting during the transmission and, therefore, decreases CPU load. However, it requires a support from operating system and a number of corresponding system calls. A more performance emphasized approach is to include RTIA right into the federation process as the additional thread. Thereby, libRTI and RTIA will automatically share the same address space. Thread-level data exchange can be more effective than the process-level one. Moreover, modern multi-core CPUs are able to provide multi-threaded execution with the additional performance gain. Unfortunately, integration of libRTI and RTIA in a single process breaks the simulation security and reliability. Still, this solution fits the purposes of HILS development and seems to be a preferable option.

Although it is not obvious at the first glance, both considered approaches require serious modification of RTIA process. RTIA is a single-thread process, and it has to wait both RTIG and libRTI messages simultaneously. There are two well-known paradigms of its implementation, namely, polled and related waiting. During the polled waiting, process just looks for incoming messages in a cycle. This requires additional CPU time. During the related waiting process asks the system to notify it when the message comes, and suspends until reception of this notification. This approach is more complicated and its efficiency is inversely proportioned to the frequency of incoming messages. RTIA receives message rare enough, thus CERTI uses related waiting. However, there is no standard way to implement related waiting of the socket and either shared memory or thread using the simple single-threaded process. Thereby, any of them requires RTIA to use polling or implement related waiting using multiple threads.

V. CONCLUSION

According to the conducted performance benchmarking, out-of-the-box CERTI RTI lags far behind the HILS STAND. Although CERTI cannot be used as a runtime for the same range of simulations, the absolute values of its latency and throughput are acceptable for the average HILS model. Therefore, the performance gap between two systems is not a critical one and can be further reduced after implementation of the stated proposals.

ACKNOWLEDGMENT

Special thanks to Dmitry Volkanov for reading the paper and giving many helpful remarks.

REFERENCES

[1] V. V. Balashov, A. G. Bakhmurov, M. V. Chistolinov, R. L. Smeliansky, D. Yu. Volkanov, N. V. Youshchenko, “A hardware-in-the-loop simulation environment for real-time systems development and architecture evaluation,” International Journal of Critical Computer-Based Systems (IJCCBS), vol. 1 - issue 1/2/3, 2010.

[2] A. Bakhmurov, A. Kapitonova, R. Smeliansky, “DYANA: An Environment for Embedded System Design and Analysis,” in Proceedings of 5-th International Conference TACAS'99, Amsterdam, The Netherlands, March 22-28, 1999. pp.390-404.

[3] R.L. Smeliansky, Yu. V. Bakalov, “A Language for Specifying Distributed Programm Behavior,” Proceedings of the VII. International Workshop on Parallel Processing by Cellular Automata and Arrays, Parcella ’96, Berlin, 1996, pp.85-92.

[4] R.E. Nance, “A history of discrete event simulation programming languages,” Blacksburg, USA, 1993.

[5] M. Adelantado, P. Siron, and Chaudron J.B., "Towards an HLA Run-time Infrastructure with Hard Real-time Capabilities," in Proceedings of International Simulation Multi-Conference, Ottava, Canada, 2010.

[6] Chemeritskiy E.V., Savenkov K.O. Towards a real-time simulation environment on the edge of current trends // In Proceedings of the 5-th Spring/Summer Young Researchers' Colloquium on Software Engeneering, SYRCoSE-2011, Yekaterinburg, Russia, may 12-13 2011, pp. 128-133.

[7] R.G. Sargent, "Requirements of a Modeling Paradigm,", Winter Simulation Conference WSC’92, Arlington, USA, 1992, pp. 780- 782.

[8] Richard D. Fujimoto, “Parallel and Distributed simulation systems,” 2000.

[9] IEEE Std 1516-2010, “IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification,” 2010.

[10] E. Noulard, J.Y. Rousselot, and P. Siron, “CERTI, an Open Source RTI, why and how,” Spring Simulation Interoperability Workshop, San Diego, USA, 2009.

[11] P. Bieber, D. Raujol, and P. Siron, “Security Architecture for Federated Cooperative Information Systems,” Annual Computer Security Applications Conference, New Orleans, USA, 2000.

[12] M. Adelantado, J.L. Bussenot, J.Y. Rousselot, P. Siron, M. Betoule, “HP-CERTI: Towards a high Performance, high Availability Open Source RTI for Composable Simulations,” Fall simulation interoperability workshop, Orlando, USA, 2004.

[13] B. d’Ausbourg, P. Siron, and E. Noulard, “Running Real Time Distributed Simulations under Linux and CERTI,” European Simulation Interoperability Workshop, Edimburgh, Scotland, 2008.

[14] L. Malinga and WH. Le Roux, “HLA RTI Performance Evaluation,” European Simulation Interoperability Workshop, Istanbul, Turkey, 2009, pp. 1-6.

[15] M. Karlsson, P. Karlsson., “An In-Depth Look at RTI Process Models,” in Proceedings of 2003 Spring Simulation Interoperability Workshop. 03S-SIW-055, 2003.

The Spruce System: quality verification of Linux file systems drivers

Karen Tsirunyan
Russian-Armenian (Slavonic) University, RAU
Yerevan, Armenia
ktsirunyan@gmail.com

Vahram Martirosyan
Russian-Armenian (Slavonic) University, RAU
Yerevan, Armenia
vmartirosyan@gmail.com

Andrey Tsyvarev
Institute for System Programming of RAS
Moscow, Russia
tsyvarev@ispras.ru [1]

***Abstract* — This paper is dedicated to the problem of dynamic verification of Linux file system drivers. Alongside with some existing solutions, the Spruce system is presented, which is dedicated to verification of drivers of certain Linux file systems. This system is being developed in the System Programming Laboratory of Russian-Armenian (Slavonic) University in Armenia. Spruce provides a large variety of tests for file system drivers. These tests help not only verify the file system functionality, but also watch the behavior of the driver in case of system failures and in rare paths.**

***Keywords* - Linux, file system, driver, Spruce, KEDR, fault simulation.**

I. Introduction

Linux-based operating systems are widely used all over the world. Linux supports several file systems, including not only own file systems (ext2, ext3, ext4) and those of other Unix-like OSs (XFS, BtrFS, JFS), but also MS Windows file systems (FAT32, NTFS). As Linux is a Free Software, it is not developed by a specific company or a developer, but rather by a community of developers, which includes thousands of system programmers. Each kernel module is developed by a certain programming team, which is responsible for its technical support. This brings us to the situation where there is no centralized quality control of source code. This makes it possible for some kinds of errors to appear in Linux. For example, no one performs integrity verification of the system in its entirety. That can usually be avoided in other operating systems, which are developed and maintained by certain companies.

File system modules are one of the most widely used components of Linux. Each Linux user needs these modules to operate correctly. Errors in file system drivers can lead to serious consequences - from data distortion and loss to critical security vulnerabilities. The change log of Linux kernel [1, 6] shows that errors in file system modules are still quite common.

Linux is a monolithic OS kernel with support of dynamically loadable modules (Loadable Kernel Modules – LKM). Usually there can be hundreds of threads working concurrently in Linux. It leads to the well-known problems of race conditions. Besides, there are no protection mechanisms between modules and the kernel itself. All the code is being executed in privileged mode which allows any of the modules to have direct access to all the others modules and the kernel. Also there is no garbage collector in Linux. All these factors make Linux module development a very hard process from the point of view of correctness. It is really easy to make mistakes in such situations but it is really hard to find those mistakes later.

Linux file system modules usually consist of two parts: common functionality and FS-specific part. The common functionality implements the back end of general-purpose system calls while the FS-specific part implements such operations as defragmentation (on-line or off-line), partition resize, migration, etc. Each of these parts in its turn consists of two parts: normal execution and error checking and handling.

Because of the complexity of Linux kernel modules there are several kinds of critical errors usually found there. First of all there can be fatal errors, which make the module no longer operational (e.g. dereferencing a null pointer). These errors usually occur because of low quality code; for example, when the error checking and handling is not there.

There are certain resources (such as memory or system objects) which need to be manually returned to the system; otherwise a leak of resources in the kernel modules is said to have occurred. The reason for the occurrence of such errors is that Linux is developed in “C” programming language which does not have its own garbage collector. On the other hand the kernel itself is not responsible for freeing resources after the module is unloaded. Next, as there are multiple threads working in the kernel concurrently, the race conditions are usual. It is one of the hardest errors to discover in kernel modules. This problem is troublesome even in user-space.

Also, there is another kind of errors in kernel modules which probably are the easiest to find and correct – incompliance with the documentation.

1 This work is partially supported by RFBR 10-07-1047a, 11-07-12075-ofi-m, Minobrnauki RF 07.514.11.410.

Testing is one of the existing methods of maintaining the desired quality level of software components. Naturally, the kernel itself and its parts are tested prior to the release, but these tests cover only the basic functionality. A system which allows to check a file system driver more thoroughly and to reveal above errors would be useful for quality assurance of Linux-based OS. This system would be used both by developers and maintainers of existing file system drivers, and by developers of the new ones.

II. Overview

Let us briefly present some of the most popular Linux testing systems.

Autotest [7] is a framework for fully automated testing. It is designed primarily to test the Linux kernel, though it is useful for many other purposes such as qualifying new hardware, virtualization testing, and other general user space program testing under Linux platforms. It's an open-source project under the GPL and is used and developed by a number of organizations, including Google, IBM, Red Hat, and many others.

The Linux Test Project (LTP) [8] is a joint project started by SGI TM and maintained by IBMR, that has a goal to deliver test suites to the open source community that validate the reliability, robustness, and stability of Linux. The LTP test suite contains a collection of tools for testing the Linux kernel and related features.

The Phoronix Test Suite (PTS) [9] is the most comprehensive testing and benchmarking platform available that provides an extensible framework for which new tests can be easily added. The software is designed to effectively carry out both qualitative and quantitative benchmarks in a clean, reproducible, and easy-to-use manner.

All of the testing systems mentioned above cover only that part of file system drivers, which is responsible for normal execution. These systems do not test the driver behavior in case of system failure and other rare execution paths.

Along with the testing systems there are certification systems developed by some major GNU/Linux distribution companies. These systems usually check complete GNU/Linux distributions for compatibility with hardware. There are certification systems developed by ***Novell, Red Hat, Oracle, Canonical, Google***. Because operations with hardware are performed via drivers corresponding to this hardware, these systems also check drivers.

Most certification systems for Linux simply use testing for verification of hardware. They use external test suites, including Autotest, LTP and PTS, and test suites specially developed for a specific system. So, such systems suffer from the same problems as described above for test suites.

But some certification systems pay more attention to checking device drivers.

While testing storage hardware, ***SUSE Yes Certified Program***[12] examines the work of its driver for memory leaks (chunks of memory which have been requested by the driver, but have not been freed by it) and accesses memory areas outside of allocated ones. Also, some tests are performed in a mode in which some memory requests from the driver may return failure. Such tests verify the operability of the driver in case of memory pressure.

For checks and for memory pressure emulation instrumentation of driver object file is used. Allocation/deallocation function calls are replaced with calls of special stubs.

Certification program ***Oracle Linux Test (OLT)***[13], beside testing of OS in normal conditions, also performs testing in conditions of system-wide memory pressure.

The checking of internal properties of driver work during testing gives much for quality driver verification in comparison with testing only.

For example, the rate of leaked memory per one device operation may be low. So, for revealing a leak, a normal test should perform many device operations before the total memory leak becomes sufficient for making the test fail. Indeed, simply increasing the number of device operations may be insufficient for triggering test failure, because not all operations may cause memory leaks (and even not in any condition).

Checking memory leaks while the driver is working changes the situation dramatically. In that case a single operation is guaranteed to be sufficient for revealing memory leaks. Moreover, error reporting in that case becomes more informative than that in case of test failure caused by memory exhaustion.

In SUSE Yes Certified program internal properties of driver work are checked for memory leaks and write-past-end/write-before-begin errors. The disadvantages of the implementation are: small number of intercepted allocation/deallocation functions (this leads to missing memory leaks) and inability to reuse implementation of those checks separately from the certified program itself.

Testing under memory pressure also improves the quality of driver verification, allowing triggering driver code which is responsible for error-processing. This, in turn, allows for verification for otherwise unexecuted code.

Linux kernel and kernel modules, which implement drivers, request memory from a common pool, and are strongly interconnected with one another.

Because of this, system-wide memory restriction applied by Oracle Linux Test is not very effective for single driver testing – there is no guarantee that concrete failure in memory allocation affects the given driver.

Memory restriction simulation based on making only those requests fail, which are performed by the driver, is much more effective. Every such failed request affects the guarantees of driver execution.

There are the following disadvantages of such simulation implementation in SUSE Yes Certified program: restriction in simulation scenarios choice (only scenarios based on random generator are available), and again inability to reuse implementation separately from the certified program itself.

The certification systems mentioned above are summarized in Table1.

TABLE 1.

	Whole modules checking	**Tests**
Suse	Exists	Own
Red Hat	not exists	Own
Oracle	Only out-of-memory imitation in a whole system	Own
Canonical (Ubuntu)	not exists	mainly external(LTP, Phoronix, ...) + own shell
Google (Chrome OS)	not exists	own + external(LTP, Autotest, Unixbench, ...)

There are also systems for dynamic analysis of Linux kernel.

Kmemleak is a memory leak detector included in the Linux kernel. It provides a way of detecting possible kernel memory leaks in a way similar to a tracing garbage collector with the difference that the orphan objects are not freed but only reported via /sys/kernel/debug/kmemleak.

Kmemcheck will trap every read and write to memory that was allocated dynamically (i.e. with kmalloc()). If a memory address is read that has not previously been written to, a message is printed to the kernel log. Kmemcheck is also part of Linux kernel.

Fault Injection Framework which is included in Linux kernel allows for infusing errors and exceptions into an application's logic to achieve a higher coverage and fault tolerance of the system.

KEDR Framework [5] is an extensible system for dynamic analysis of kernel modules (device drivers, file system modules, etc.) in Linux on x86 systems. KEDR tools operate on the modules chosen by the user and can detect memory leaks, perform fault simulation as well as other kinds of data collection and analysis. KEDR-based tools have already proven their effectiveness by finding errors in several widely used kernel modules [3]. KEDR framework is Free Software and is distributed under the terms of GNU General Public License Version 2.

All of these tools have different abilities. None of them is strictly superior to the others. The advantages of the KEDR Framework include the following:

- Operation with a specific module.
  On execution KEDR waits for the target module to be loaded into the kernel. On module load KEDR finds out the calls of the kernel functions from the target module and replaces them with calls to the corresponding functions from the KEDR framework.
- Possibility to define well-tuned scenarios of fault simulation.
  KEDR tools support configuration files which can define some specific conditions for the fault simulation to be activated. For example, it can simulate failures for a single kernel function with some probability or exact frequency or even when some conditions on function arguments are met.
- Possibility to extend the functionality. KEDR framework and KEDR Tools can be extended by new call monitors, fault simulators, trace analyzers etc.

Our goal is to verify both the normal functioning of the drivers and their behavior in critical situations. The revealing of driver errors is vital, insofar as a single error in the driver can result in the failure of the whole system. Additionally, approximately half of errors found in the kernel are statistically in the drivers and file systems [9].

III. SPRUCE SYSTEM

All the testing systems mentioned above are operating only with the part of normal execution of FS modules (they verify the kernel drivers' functionality according to documentation).

One of the most important parts of the drivers – the one that is not covered by existing systems – is the error checking and handling. It is clear that any function call (in our case kernel functions) can fail. For example, there can be lack of resources (memory, internal kernel objects) which makes resource allocation functions fail. In theory, everyone should check the return values of all the functions called in code. In practice, however, developers often forget to add checks and handle error cases. This is very dangerous in case of drivers, since an unhandled error can result in the corresponding driver module becoming unloadable or disabled. This, in turn, will lead to undefined behavior of the corresponding device.

It is therefore very important to be certain that the file system driver handles all possible errors and faults.

The Spruce project [2, 3] is designed to verify several Linux file system drivers, including Ext4, BtrFS, XFS, JFS. The system consists of several modules.

Every module has two execution scenarios – normal execution and fault simulation. In normal execution mode, the module verifies the functionality of the driver according to the documentation. In this case each error in the driver is deemed as a test failure. This case is more or less included in existing testing systems.

The main advantage of the Spruce system is that it can also cover the error checking and handling part of the code, which comprises about one third of the whole source code.

Below is the description of the Spruce system modules:

- ***Main module***. Implements the user interface. Allows to define some configurable values (which modules should be executed, which file system drivers should be checked, where to store the execution log).
- ***System call checker***. This is the module which provides the major part of the code coverage. It verifies the system calls which concern to the file systems such as *creat, open, fsync*. The verification is done based on the POSIX manual pages. This module covers almost all the code except some of the error handling lines. (Only with normal execution scenario).
- ***Common operations***. This module checks the functionality of the common system utilities which concern to the file systems such as *cp, mkdir, ln*.
- ***Benchmark***. This module provides benchmark testing of a wide variety of operations including creating and removing large files, compression and decompression of files, reading and writing large amount of data etc.
- ***File system specific modules***. This is not just a module but a set of modules, which cover all the FS-specific code in the corresponding kernel modules. The specific features include online resize, online defragmentation, delayed allocation, migration from other file systems.

The Spruce system is implemented mainly in C++ language using object-oriented design. Each test is executed in its own process. It makes the whole system much more stable. If any of the tests crashes or runs for too long it will not affect other tests. The parent process takes care of the test.

Each module can be configured to be executed in different ways. For instance, the user can decide which system calls should be verified, or which test should be performed.

In an ordinary environment, it is almost impossible to simulate fault situations, error paths, rare execution paths etc. In order to solve this problem the KEDR framework is to be used.

There are of course several tools which could also be used to achieve this goal. They are KmemLeak, KmemCheck and Fault Injection tools provided by the Linux kernel itself. Let's see why KEDR framework is preferable to these tools. Verification of a kernel module (in our case a single file system driver) must be done in separation from the other parts of the system. This means that if something must be modified in the system it had better concern only the module to be verified.

Except for the module-based issues, Spruce system needs some mechanism to make the file system driver execute all the rare execution paths and error handling parts of code. This means that there is a need for some kernel functions to fail under certain conditions. For example, to make the file system driver execute all its memory allocation error handlers, it is not enough to make all the calls of *kmalloc* (and similar functions) to fail. In that case usually only the first error handling code would be activated. That is because such errors (memory allocation failures) make the driver's current function execution impossible. Such error handlers usually stop the function execution, returning some error code to the user.

Besides the memory allocation failures there can be other kinds of kernel functions which would also need to be simulated. This means that there can be need to extend the set of the supported functions.

It is usual for Linux file system drivers to check for some capabilities prior to the operation execution. That is why some parts of the file system driver source code are really rarely executed, because users usually have some basic capabilities. So Spruce needs the corresponding kernel function (*capable*) also to be simulated.

The list of the necessary functionality is quite similar to the feature list of the KEDR Framework and KEDR-based Tools. It makes KEDR really suitable for the Spruce system. On the other hand, all the other kernel module analyzing tools provided by the Linux kernel cover only some of the presented needs. That's why KEDR was chosen as a supporting framework for the Spruce system.

With KEDR, one can artificially simulate memory allocation errors (and potentially any other errors in kernel functions used by the driver). KEDR can be configured to make the driver pass through all errors which would be impossible in normal execution. Nevertheless, in normal execution mode, the Spruce system cannot cover more than 70% of file system driver code. This is because in any well-designed and well-implemented system, approximately one third of the code is dedicated to error checking and handling. However, the system call testing module covers some error cases, since it analyzes a number of argument value sets for system calls. So with KEDR-based tools Spruce system could be able to cover also part of the remaining 30% of code.

After analyzing all the above mentioned aspects of file system driver verification, the quality verification system can be defined. Such a system must make the driver pass through all the possible execution paths (even those not developed in the source code). This means the following:

- Make the driver perform all the normal execution paths.
- Make the driver operate in out-of-resources and other faulty situations to make sure that the driver does not fail.
- Make the driver confront some really rare situations during the execution.

So, our purpose is to develop a verification system which can test several Linux file system drivers in scenarios mentioned above. That can be achieved by testing all the possible use cases of the drivers (according to the documentation) and using the KEDR framework and KEDR tools (if necessary extending the KEDR tool set).

It is clear that if a verification system covers only some parts of the driver source code, missing such important parts as error checking/handling and rarely executed code, it does not qualify to be a high quality verification system. On the other hand a 100% code coverage does not necessarily mean high quality verification. There can be pieces of code which ***should be there*** but are missing. For example, if the driver does not check the status code returned by a function, even a 100% code coverage cannot find out that error. To reveal such errors the testing system should do something more: make the called function fail. None of the mentioned testing systems are able to do such a thing.

Still, it can be stated that (under even conditions) the verification system which brings to higher percent of code coverage is better than the others.

IV. Current State

As of now, the modules ***Main, Benchmark,*** and ***Syscall*** have been implemented (normal execution mode only). Verification methods for FS-specific drivers capabilities are being investigated.

Code coverage analysis has been performed to calculate the Spruce system quality. That could show how the Spruce system is competitive with the existing Linux testing systems.

Table 2 and Table 3 present the coverage values for drivers of several file systems in kernel version 3.2.9 according to the execution of above mentioned verification systems and the Spruce system. The data has been acquired by means of Gcov tool [11], which is a part of GCC.

TABLE 2.

OS x64	**Ext4**	**Btrfs**	**Xfs**
LTP	40.1%	42.9%	42.9%
PTS	34.6.%	36.2%	32.3%
Spruce	40.9%	36.8%	44.0%

TABLE 3.

OS x86	**Ext4**	**Btrfs**	**Xfs**
LTP	42.8%	39.2%	39.3%
PTS	34.5%	35.7%	32.5%
Spruce	40.7%	35.4%	40.8%

The figures in tables show that even the partially developed Spruce system is already competitive with the existing solutions (coverage analysis for Autotest is not done because our goal was not to get those values but to be able to compare Spruce with some of the leading testing systems). Moreover, in its current state, the Spruce system execution takes less than 4 seconds. For a sample comparison, it takes LTP two minutes. Of course the figures are acquired by running the systems under the same conditions, i.e. LTP is executed only on those tests which check those and only those system calls that Spruce does. Also in those conditions LTP gives only 34% code coverage, when Spruce gives 41%.

V. Future Directions

We are planning on implementing the incomplete modules of Spruce system. It is also planned to utilize the KEDR-based tools for fault simulation scenarios in those kernel functions which are often used in file system drivers. If necessary, we will upgrade KEDR to provide broader functionality convenient for the Spruce system. It will allow for gaining higher quality of Linux file system drivers verification using the Spruce system.

VI. Conclusion

In this paper we have analyzed the errors which usually occur in Linux kernel modules and especially in file system drivers. Later several testing and certification systems were presented and analyzed to find out in what way and how well they perform verification. Also for that reason the source code coverage was calculated for several file system drivers. Then the Spruce system was presented which is designed to perform high quality verification of Linux file system drivers. The goal is to be achieved using the KEDR framework and tools.

References

[1] A.V. Khoroshilov, V.S. Mutilin, E.M. Novikov, P.E. Shved, A.V. Strakh. Linux Driver Verification Architecture. Proceedings of the Institute for System Programming of RAS, volume 20, 2011 г. ISSN 2220-6426 (Online).

[2] Martirosyan V., Shatokhin E., Gishyan S. Dynamic verification of linux file system drivers. Proceedings of Computer Science and Information Technologies conference, Yerevan, 2011.

[3] Rubanov V., Shatokhin E.. Runtime Verification of Linux Kernel Modules Based on Call Interception. Proceedings of IEEE International Conference on Software Testing, Verification and Validation (ICST'11), Berlin, Germany, March 2011.

[4] Spruce system, https://code.google.com/p/spruce.

[5] KEDR framework, http://code.google.com/p/kedr/

[6] The Linux Kernel Repository - Change Logs, http://git.kernel.org/?p=linux/kernel/git/torvalds/linux-2.6.git

[7] Autotest Framework, http://autotest.kernel.org.

[8] Linux Test Project, http://ltp.sourceforge.net.

[9] Phoronix Test Suite, http://www.phoronix-test-suite.com.

[10] Linux Kernel Bugzilla, https://bugzilla.kernel.org

[11] Project Gcov, http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

[12] Storage Test Tools for SUSE YES Certified Program, http://www.novell.com/developer/ndk/storage_test_tools.html

[13] Oracle Linux Test Project, http://oss.oracle.com/projects/olt/

Deterministic replay of program execution based on Valgrind framework.

Research-in-progress report
Maksim Ryndin
Moscow Institute of Physics and Technology, Dolgoprudny, Russian Federation
Scientific supervisor: S. S. Gaisaryan, Institute for System Programming, Moscow, Russian Federation

***Abstract*—Deterministic replay is execution of the same sequence of instructions with the same arguments as at the first run.**

Developers spend a lot of time in trying to fix bugs in their software. Sometimes it is very difficult even to reproduce the bug. This often occurs in multithreaded applications. According to NIST[1] errors in software are estimated $59.5 billion annually. Therefore deterministic replay of program execution has become a problem of interests of many researches recently. Deterministic replay could help to solve the problem of bug reproduction and provide more abilities for program analysis.

Valgrind is a framework for dynamic binary analysis. It provides infrastructure for translation and instrumentation of executable code.

The goal of the work is to create a tool providing deterministic replay of program execution based on Valgrind infrastructure.

I. Introduction

Application programs are widely used in modern world: for automation of business processes, for scientific calculations, for processing large amount of data, etc. The programs become more and more complex and large to provide all needs of people they are used by. The larger the program the more difficult to implement it without errors. So bugs occur quite often and it is necessary to be able to find and fix them quickly.

Finding of bugs' causes could be complicated by unreproducible behavior. Such errors are the most painful for developers and can take a lot of time to fix them.

Multi-core processors become the norm recently. So there is an interest in writing parallel programs. Such programs are more efficient for wide class of applications. Threads in one program must be synchronized in a proper way to prevent race conditions. But it is difficult to guarantee correct synchronization and threads in real programs behave differently from run to run. Often such differences are invisible and never cause errors. But sometimes they may cause or may not cause an error depending of some circumstances. In that case it becomes unreproducible bug.

By deterministic replay one should mean execution of the same sequence of instructions with the same arguments as at the first run. Deterministic replay would provide ability to reproduce bugs at each run.

Several generic dynamic binary instrumentation frameworks exist, such as Pin[2], DynamoRIO[3] and Valgrind[4][5]. These tools make possible to detect variety of errors. E.g. Valgrind distribution includes the following debugging and profiling tools: Memcheck, Cachegrind, Massif, Helgrind, DRD[6]. Memcheck detects wrong memory accesses (e.g. when accessed region of memory is not allocated), usage of uninitialised values, memory leakages, etc. Cachegrind profiles the cache, Massif profiles the heap. Helgrind and DRD both detect race conditions with some differences.

Pin is a proprietary program and it is free for non-commercial use only [7]. Copyright restrictions may also complicate code modifications for research goals.

All these frameworks, however, do not provide deterministic replay of program execution.

BugNet[8][9] and PinPlay[10] both provide deterministic replay. However, at the moment of writing this paper author couldn't find neither source code nor executables of any of them. PinPlay is a proprietary program too, so it would have all license limitations of Pin.

II. Existing tools

Several research papers describing architecture of BugNet and PinPlay are available.

A. BugNet

BugNet is based on the observation that the following information is sufficient to replay a program's execution in a deterministic way: initial architecture state (program counter and register values), architecture state updates from system calls and interrupts and used load values[8].

Architecture state can be kept in a consistent state by recording of register values and program counter after servicing system calls and interruptions.

Logging of each memory load causes overheads, so BugNet logs only the first access to a memory location. To accomplish that the tool marks logged memory addresses. Taking into account the external updates of logged memory such as system calls, DMA and shared-memory interactions, BugNet unmarks corresponding location's address when such events occur. Than it logs again when the location is accessed next time.

BugNet breaks a thread's execution into checkpoint intervals. For a checkpoint interval BugNet stores enough information to start replaying program's execution from the start of the checkpoint. An interval is terminated by system call, interrupt, context switch or when size of stored data exceeds some predefined value.

In addition to data, BugNet records information about the code executed during logging. The information contains name and path of the binary or library loaded, a checksum to represent the version of the binary or library and the starting address where it was loaded. This information goes to *code log*.

Replay of a checkpoint interval starts with reading the code log and restoring of code space. Then architecture state (program counter and register values) is set. Then executions starts. For every load instruction the replayer obtains the load value from load-log if the value was not obtained earlier. At the end of the checkpoint interval a new one starts.

B. PinPlay

PinPlay consists of two Pintools[10]: a *logger* and a *replayer*. The logger stores information about program execution in a set of files called *pinball*. The replayer repeats program execution using information from pinball.

A very powerful feature of PinPlay is ability to combine the logger and replayer with most existing Pintools and GDB. It gives lots of useful information about a bug and helps fixing it easily.

The PinPlay logger stores only minimal information necessary to reproduce the non-deterministic events and thus does not generate too large pinballs. The sources of non-determinism are:

- *Initial stack location*: the location is assigned by the kernel and may differ from run to run
- *Data location changes*: addresses of allocated memory may also differ
- *Program code changes*: code of shared libraries may change from machine to machine
- *CPU specific instruction behavior*
- *Signals*: signals are delivered by the kernel, so they are not guarenteed to arrive at the same execution point
- *Uninitialized memory reads*
- *Behavior on system calls*: it may change over time, e.g. *gettimeofday()*
- *Behavior on shared memory accesses*

The approaches to handle these sources used in PinPlay are:

- PinPlay logs addresses of the memory ranges in use and then preallocates them before replaying
- Code of shared libraries is captured during logging and restored during replay
- Log changes in registers' values after CPU-specific instructions
- Time of signal arrival is logged in terms of instruction count since beginning of execution
- PinPlay skips most of the system calls and restores architecture state after each of them during replay

III. Valgrind overview

Valgrind is an instrumentation framework for building dynamic analysis tools. The program distribution includes six tools, one can also create a new one. Valgrind is licensed under the GNU General Public License, version 2.

The key concept of the framework's architecture is the division between its *core* and *tools*.

The core does low-level work for program instrumentation. It contains: the JIT compiler, low-level memory manager, signal handling unit and a thread scheduler. It also performs system calls for the tool.

The tool is responsible for program instrumentation. Valgrind provides tool programming interface: functions to be called when certain event of interest occurs. The arguments of the functions contain sensible information about the event.

The Valgrind translates code blocks on demand. The translation unit is a block of code ending with one the followings: an instruction limit is reached, a conditional branch is hit, branch to an unknown target is hit. The code translation may be splitted into several phases:

- *Disassembly*: converting machine code into intermediate representation (IR) tree
- *Optimisation 1*: flattening the tree IR, copy and constant propagation, redundant code elimination
- *Instrumentation*: the code block is passed to the tool which transforms it (except the Nullgrind – the tool which does not instrument anything)
- *Optimisation 2*: constant folding and dead code removal
- *Tree building*
- *Instruction selection*: converting tree IR into a list of instructions which use virtual registers
- *Register allocation*: replacing virtual registers with host registers
- *Assembly*: encoding the selected instructions into executable code and writing it to a block of memory

IV. Implementation progress

The key idea of implementation is to use existing tool programming interface. Not only it gives an ability to track events such as system calls and signal arrivals, but also *event-aggregators* such as memory reads (which can be caused by different system calls). This significantly simplifies logging of memory accesses. At the moment regrind (tool for deterministic replay) prototype can work in two modes: *prepare-replay-mode* and *perform-replay-mode*.

In the first mode regrind tracks following events:

- Signal arrival
  1) Before arrival
  2) After arrival
- System calls
  1) Before system call
  2) After system call
- Accesses to memory:
  1) Before memory reads
  2) After memory write

The first two types of events are tracked only for statistic collecting at the moment. But as for the third type of events – accesses to memory – the tool prototype also records accessed memory regions.

Every log point starts with an instruction counter which equals the number of executed guest instructions. “Guest” means that only instructions of program in question are taken into account, not those added by Valgrind. At the moment the log file is written in plain text to simplify debugging.

In the perform-replay-mode regrind restores data about memory accesses. When a memory read occurs regrind gives value from the log file to the tracker functions and thus substitues current value of the memory.

The current implementation does not guarantee exact the same instruction sequence yet but makes visible behavior of some simple programs the same from run to run.

Let’s consider an example: linux command *date*.

At first regrind runs in a prepare-replay-mode. The events of interest are memory reads and memory writes. Information about one of them looks like:

```
4e83a:pre_mem_read:tid[1]
:base[4025000]:size[1d]
Fri Mar 30 23:54:50 MSK 2012
```

The first number `4e83a` is an instruction counter. `pre_mem_read` indicates the type of the event. Then arguments of the callback follow: `tid` is thread identifier, `base` is address of memory range and `size` is it’s size. Content of the memory region (dump) is presented at the next line of the log file.

In perform-replay-mode regrind executes instrumented guest code. If one of tracked events occurs the tool reads the next checkpoint from the log file, allocates memory and fill it with dump content. Valgrind’s core passes arguments to the tool: thread identifier, address and size of the accessed region. Address and size are replaced in the tool with new values.

Thus printed value of the command date will not be equal the actual date and time, but it will be the same as it was at the first run.

Valgrind provides integration with GDB. One can monitor execution of instrumented program. So at the moment it is possible to use both regrind and GDB to perform analysis of reproduced exectuion.

V. Future research

The log file in plain text format is large. It is planned to log data in a binary form to decrease this overhead and provide a simple tool to convert it to a human readable format.

At the moment the regrind prototype does not skip system calls with known results. It is useless work to perform such system calls, so they are to be eliminated in the final version of the tool.

It is also planned to use more information about non-determinism:

- Deliver signals at the same moment as they were delivered at the first run
- Preallocate stack location and used memory regions

Intergation of regrind and GDB allows analyze non-reproducible bugs more efficiently. But integration of regrind with other Valgrind tools will give more powerfull abilities. So it is planned to be implemented in the future too.

VI. Conclusion

The aim of the work is to provide open source tool for deterministic replay of program execution. Existing tools solving the problem are proprietary and not fully available.

The key concept is to use Valgrind – a framework for dynamic binary instrumentation. The framework provides tool programming interface which makes possible to track different events during program execution.

The current implementation reproduces execution of some programs in a seemingly deterministic way (print out does not change from run to run). Some features in the tool are planned to be implemented. The features include covering more sources of non-determinism, improvements of tool architecture and integration with other Valgrind tools.

References

[1] G. Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, National Institute for Standards Technology, 2002

[2] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallance, V. J. Reddi, K. Hazelwood, *Pin: Building Customized Program Analysis Tools With Dymanic Instrumentation*

[3] D. Bruening, T. Garnett, S. Amarasinghe, *An Infrastructure for Adaptive Dynamic Optimization*, Proceedings of CGO’03, San Francisco, USA, 2003

[4] N. Nethercote, *Dynamic Binary Analysis and Instrumentation*, PhD thesis, University of Cambridge, UK, 2004

[5] N. Nethercote, J. Seward, *Valgrind: A framework for Heavyweight Dynamic Binary Instrumentation*, 2007

[6] http://valgrind.org/

[7] http://pintool.org/

[8] S. Narayanasamy, G. Pokam, B. Calder, *BugNet: Recording Application-Level Execution for Deterministic Replay Debugging*, University of California, USA, 2006

[9] S. Narayanasamy, G. Pokam, B. Calder, *Software Profiling for Deterministic Replay Debugging of User Code*, University of California, USA

[10] H. Patil, C. Pereira, M. Stallcup, G. Lueck, J. Cownie, *PinPlay: A Framework for Deterministic Replay and Reproducible Analysis of Parallel Programs*

Simulation Analysis Framework Based on TRIAD.NET

Grigorii Kolevatov
Department of Mechanics and Mathematics
Perm State University
Perm
kolevatov@prognoz

Elena Zamyatina
Department of Mechanics and Mathematics
Perm State University
Perm
e_zamyatina@mail.ru

Abstract — **the objective of using simulations is to produce true knowledge about complex dynamic systems. Due to increasing popularity of simulation, complexity of simulation models is increasing too. In recent years there were researches with simulations which consist of thousands of interacting objects. The process of analysis such models becomes too complicated and makes the whole modeling process more expensive. That influence tends to be very strong if, as usual, an iteration process is used. Moreover, simulation output analysis based on number of math statistics techniques which creates extra requirements to analysis team members and makes simulation too difficult to mass implementing. This paper is dedicated to attempt of the problem resolving.**

simulation, simulation output analysis, data mining, time series, linear regression

I. Introduction

Computer simulations have many applications in many areas, such as: logistic, manufacturing, medicine and service operations. Simulation is used when traditional Operations Research tools such as linear programming, stochastic modeling or queuing network models cannot capture the details or the dynamic of the system [1].

Although simulation is good for representing complex systems, analysis of simulation seems to be too complicated. As shown in paper [2] traditional simulation analysis looks like selection the best variant by some criteria from different scenarios. Setting the set of scenarios and selection criteria has a major influence on analysis results. Decision, which criteria is most appropriate and what scenarios need to be considered, is often made on basis of intuition and modeler experience. That form of analysis does not help researcher to understand why the selected scenario is the best and if the number of scenarios is enough or not.

On the other hand, in last three decades, a huge number of data analysis techniques called Data Mining have been developed. Application of those techniques to the problem of simulation output analysis could significantly decrease analysis complexity and improves real profit of simulation applications.

Many attempts to solve the problem were made in recent scientific researches. Paper [3] shows the benefits of using heuristic-based automated optimizations for finding better scenario with pre-set selection criteria, called objective functions. New approach to automate selection criteria was described in [4]. Offered approach helps to reduce the requirements to the modeler due to automating statistic's calculations. In [1] the way of reducing simulation output log by mining information about entire model variables relationships were demonstrated. Number of data mining techniques, which was applied in [1], estimate the correlation between model variables. The approach was tested on complex model with huge number of interacting elements. It earnestly shows how significantly data mining applications can decrease volume of output data without losing important information.

All of described here approaches have one main disadvantage – they were designed for particular specific tasks and have not got facilities to simplify the analysis process in general case. The decision of the problem could be a special designed simulation analysis framework which could provide an expert with set of instruments which would help an expert to answer every question about simulation model that an expert could have. It should be able to adjust dynamically to particular area specifications. This framework should combine information about analyzed system (extracted from model describing) and new data mining techniques. Also it should be able to adapt dynamically to an expert needs. This paper describes the attempt of creating such framework. Also it is necessary to mention, that developing the most common framework which suits to every particular case is a very difficult task. Thus, in the paper only attempt of creating such framework is considering. Framework creating activities were undertaken on base of discrete-event simulation environment TRIAD.NET. In section II formal problem definition is described. TRIAD.NET Special features' description is put into Section III. Suggested problem solution is described in Section IV. Section V is dedicated to particular framework architecture. At the end of the paper brief conclusion is made.

II. Problem Definition

A. User Dialog

At first, it is necessary to mention that all knowledge about the real system: its purpose and its place in the world is situated in expert's mind. Modeling environment has only information about model itself. That's why it is impossible to put all of research activities into environment. Only an expert can determine what information is important: if model reflects real system in proper way, if all necessary information is delivered

or not. So, the expert should deliver to the environment necessary information about research objectives. The second assumption is that the expert's knowledge grows dynamically since research was started. It means that an expert needs system to automatically adapt to expert's requirements. By that reasons it's logically to decide that the interaction between an expert and the environment should be organized in terms of dialog: expert asks questions about his filed of interest, environment makes an investigation and returns answers [5].

In traditional analysis all questions, that analyzer could ask an environment, in general, was: "Is *A* true?" where *A* is some statement about model values. It's not enough, because usually, expert develops simulation not only for checking some suggestion, but for changing suggestion to a more appropriate value. That's why it is necessary to add another type of question: "Is *A* determined by *B*?" It means that *A* is true when *B* is true, and when *B* is false – *A* is false too. There is one more is "Is *A* conditioned by *B*?" It means that *A* is true when *B* is true, and *A* is undefined when *B* is false. To answer this question an expert usually does many different experiments with different parameters. But model parameters' domain is known to environment, model is also known. That means, consequently, environment could answer these questions itself. In fact, even answering these questions can make analysis process simpler and clearer. But also those types of question could be combined into more general: "What determine *A*" and "What condition *A*", and replaced by one: "Why *A* is true". Thus, we have two general questions:

1) Is A true?

2) Why A is true?

B. Statements.

Simulation models consist of some objects of different types and connections between them. Model variables reflect states of these objects or groups of objects or model in general. Thus, every statement about a model could be impressed by first-order predicate language. Therefore, such statements could consist of:

- the quantifier symbols $\forall$ and $\exists$;
- the logical connectives;
- parentheses, brackets, and other punctuation symbols;
- an infinite set of variables;
- an equality symbol;
- first-order predicates;
- functions.

Before saying a word about sense of those symbols we should determine that there are two types of simulations analysis: *single experiment analysis* (SEA), *multiply experiment analysis* (MEA). In different situations the particular sense of alphabet symbols should be defined in different ways.

Let *M* is a model, and *X* is a set of model variables:

$$X = \{x_1, \ldots x_n\} \tag{1}$$

In SEA case, particular value of variable x_i depends of system time and could be written as:

$$x_i : T \rightarrow D(x_i), \tag{2}$$

where *T* is system time. Hence, in SEA situation we have only one variable – *t*, and variables $x_i(t)$ would be the function of *t*.

In MEA case, value of a variable depends on experiment where it is calculated and system time. Experiment itself is determined by model parameters value and initial values for random numbers generator (RNG). To not depend from probability factors, better determine x_i in such way:

$$x_i : T \cap E \rightarrow D(x_i), \tag{3}$$

where *E* is set of all possible experiments.

Also, it's usually necessary to use some integrated characteristic, such as average or divergence. Set of those variables could be called *Y*, and every y_i could be defined in that way:

$$y_i : E \rightarrow D(x_i), \tag{4}$$

We can easily translate our question "Is *B* determine *A*" into first-order predicate form as it shown on (5) and "Is *C* determine *A*" as it shown on (6).

$$B \Leftrightarrow A \tag{5}$$

$$C \Rightarrow A \tag{6}$$

Thus, answering question "What determine *A*", we should find such statement *B* that suit next condition:

$$B \Leftrightarrow A \vee (\forall S \mid S \Leftrightarrow A) : B \vee S = 1 \tag{7}$$

Analogically, question "What condition A" could be defined as:

$$C \Rightarrow A \vee (\forall S \mid S \Rightarrow A) : C \vee S = 1 \tag{8}$$

Thus, we can define our objective as:

1) calculate value of statement A;

2) find statemnts B and C which satisfy to (7) and (8).

III. TRIAD.NET FEATURES

Distributed simulation system Triad.Net includes following components: TriadCompile – compiler from Triad modeling language, TriadCore – simulation core, GUI, TriadDebugger – validating and debugging system, TriadBalance – distributed components synchronization system, TriadEditor – remote access system, TriadSecurity – external and internal security threats detection system, TriadBuilder – automated model redefining system and TriadMining – simulation output analysis system.

Simulation model in Triad defined as:

$$M = (STR, ROUT, MES), \tag{9}$$

where *STR* – structure layer, *ROUT* – routine layer, *MES* – message layer. Structure layer represents itself as objects'

aggregate, interacting to each other sending messages. Each object has input and output poles which servers as receivers and senders messages. Structure layer representation based on graphs. Separate objects play a role of graph nodes. Graph arches determine connections between objects.

Objects' behavior is determined by routine layer. Routine is a sequence of events which plans each other. When event occurs state of associate object changes. Routine layer is separated from structure layer, thus, routines could be reused when structure is defined, and different routines could be associated with different nodes in structure layer. Message layer is used for complex message defining.

One of the advantages of Triad, which play the main role in choosing Triad as a platform for analysis framework is its special feature: special objects, called *information procedures* and *simulation conditions*. Information procedures are objects that collect information about model variables changing. When change of a model variable occur information procedure is executed and data is saved in modelling output. Specific content of data depends on an algorithm of specific information procedure. Triad has facilities to code every formal algorithm as an information procedure. Simulation conditions determine a situation when simulation ends. They also have special algorithmic faculties that help to set complicated conditions.

Simulation conditions and information procedures are separated from model definition. Hence, information procedures could be changed without model changing and the same simulation conditions could be used for different models. Algorithmic facilities of information procedures and simulation conditions and their separate (from model definition) character has not got analogues in other modelling environment and languages such as GPSS, ProModel, Witness, AnyLogic and etc. This is the main reason, why analysis framework should be developed on base of Tried.

IV. Problem Solution

As described in Section III problem consists of two parts:

1) Calculate statement value;

2) Find determining and conditioning statements.

Further, an approach of solution each part of the problem is considered.

A. Statement Value Calculation

When SEA case is considered, there are no difficulties in calculating statement value. After an experiment took place it's easy to calculate actual value of statement, based on values collected through the experiment. All model variables depend on system time, since experiment is finite then system time is finite, and we got a finite number of variables' values.

Situation becomes difficult when MEA case took place. Due to probabilistic nature of experiment, E is infinite. It makes it impossible to calculate the sentence using all of possible data. In [4] it's earnestly shown that it's simple to determine the necessary value of replications by specifying the significance level and deviation of confident interval which is suitable for situation. Back to Section II, modeling environment does not have enough knowledge about what significance level and deviation is enough to be confident in results. Hence, these parameters should be user-defined.

The second problem connected with parameters' domain. Since it can be infinite or too large it becomes too difficult to calculate all possible experiments. Problem could be solved by setting number of intervals, which could divide the domain into finite number of values.

Hence we can calculate any value of any variable, function or predicate both in SEA and MEA cases. Consequently we can calculate the actual meaning of statement.

B. Determinating and Conditioning Statement Search

The corner stone of DE terminating and conditioning statement search is to find a statement which satisfy criteria, defined in (7) or (8). These criteria could be divided into two separated criteria: dependence criteria and completeness criteria. Dependence criteria for (7) described in (5) and dependence criteria for (8) described in (6). Completeness criterion for (7) is shown in (10), and completeness criterion for (8) is shown in (11).

$$(\forall S \mid S \Leftrightarrow A) : B \vee S = 1 \quad (10)$$

$$(\forall S \mid S \Rightarrow A) : C \vee S = 1 \quad (11)$$

Total search algorithm for (7) could be described in follows way:

1) Let B be false.

2) Find new S, which suits for (5).

3) If S was found then continue, else exit.

4) Check (10) criteria.

5) If (10) criteria is false then let B = BvC, simplify B.

6) Return to step 2.

The only undefined step in these criteria is step 2. It could be divided into two steps:

1) Make new suggestion.

2) If it is impossible to make new suggestion then exit.

3) Check criteria (5).

4) Return to step 1.

Step 1 is still undefined. The way we can define it is to make father suggestion: "Variables from statement A and from statement B should have dependence". Dependence estimation is widely spread technique used in data mining problems. Different types of correlation could be estimated by special measures which are described in next section. In common way dependence estimator could be defined as it shown in (12).

$$de : X \times X \rightarrow R \quad (13)$$

Thus, dependence estimator *de* is reflecting all possible pairs of model variables into real numbers. Thus, using dependence measure of some kind, it's possible to estimate relationships between variables and choose set of variables that could be used for creating possible B-statement.

C. Dependance estimation

The objective of using dependence estimator is to estimate function dependence between variable. Consider the offered approach.

Let M – certain model. Model has parameters $P=(P_1,..., P_n)$. Each parameter is determined on domain D_{P_i} and domains organize space D_P. Model also has set of variables $X=(X_1,..., X_n)$ each variable determined on D_{X_i} what construct space D_X. Let's declare $X_i^{'}$ which satisfy the follows:

$$X_i \notin X_i^{'}, X_j \in X_i^{'}, \forall j \neq i \tag{14}$$

The result of experiment e would be matrix:

$$X^e=(X_1^e,..., X_m^e), \tag{15}$$

$$X_i^e=(x_{1,i}^e,...,x^e{}_{T,i}), \tag{16}$$

We can determine now a dependence estimation problem in such way: for the results of experiment e on model M find functions F_i which satisfy (17):

$$F_i^e : D_{A_i^{'}} \to D_{A_i}, F_i(A^{e'}) \in \left[A_i^e - \partial, A_i^e + \partial\right], \partial \in D(A_j) \tag{17}$$

Hence, dependence estimator should have a view:

$$F'=(F_1',..., F_m') \tag{18}$$

V. TRIAD.MINING ARCHITECTURE

In general, Triad.Mining algorithm is follows:

1) Get an expert request.

2) Translate request into set of information procedures and modeling conditions.

3) Simulate with conditions and information procedures from step 2.

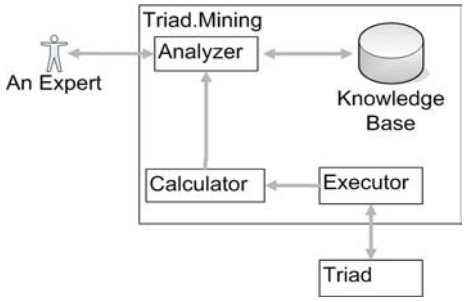
4) Process the result:

a) Calculate statement if request type is 1;

b) Start the algorithm from section IV B.

To solve the problem from section II TRIAD.Mining uses four components:

1) Analyzer
2) Executor
3) Calculator
4) Knowledge Base



Picture 1. The Architecture of Triad.Mining

Relationships between components are shown on picture 1. Analyzer gets the request from an expert in a first-order predicate form. Then it translates the request into particular modeling conditions and information procedures and sends it to the executor. Executor does simulation tasks and collects the results. Results go to Calculator. Calculator interprets the results and calculates statement value or checks the criteria. The result of the calculations goes to analyzer, which formulate the final answer to the expert.

VI. CONCLUSION

Triad.Mining shows a better result in understanding the real message of simulation output and improves efficiency of simulation research significantly in comparison with traditional simulation analysis tools. It based on common approach and does not depend on specific research area, such as logistic. It also forms up communications with and expert with and natural question-reply way, and does not demand an expert to have a deep knowledge in mathematical statistics. All of these arguments say that Triad.Mining could be used for improving efficiency of simulation research process.

[1] T. Brady, E. Yellig, Simulation Data Mining: a new form of simulation output, 37th Winter Simulation Conference, Orlando, USA, 2005, pp 285-289.

[2] Akbay, S. Kunter, Using simulation optimization to find the best solution. Industrial Engineering Solutions 28, San Fernando, Argentine, 1996, pp 24-29.

[3] Brady, Thomas F. and Bowden, Royce A. The effectiveness of generic optimization routines in computer simulation languages. In Proceedings of the 10th Industrial Engineering Research Conference, [CD-ROM], Dallas, USA, 2001.

[4] Robinson S., Automated Analysis of Simulation Output Data, proceeding of the 37th Winter Simulation Conference, Orlando, USA, 2005, pp 763-770.

[5] G. Neumann, J. Tolujew, From Tracefile Analysis to Understanding the Message of Simulation Results, proceeding of the 7th EUROSIM Congress on Modeling and Simulation, Prague, Czechia, 2010.

Meta-database for the information systems development platform

Yury Rogozov, Alexander Sviridov, Sergey Kucheov
System analysis and Telecommunications dept.
Taganrog Institute of Technology, Southern Federal University
Taganrog, Russian Federation
rogozov@tsure.ru, sviridov@tsure.ru s.a.kucherov@gmail.com

***Abstract* — For today conditions of using information systems presuppose availability of tools, by which developers can quickly adjust their products in accordance with the updated requirements. In connection with this arises the problem of automating the information systems development and reducing the share of labor expenditures for writing the source code. The paper presents a solution for this problem, which is based on the modernization of data and knowledge storing technologies. The concept of the meta-database is proposed. The requirements for the meta-database are formulated. The formal and the graphical model are given; features of the meta-database implementation with using relational technology are described. The questions of constructing and utilization a development platform based on the meta-database are examined**

Keywords – metamodel; metamodelling; meta-database; information systems development platform; automation

I. INTRODUCTION

The problem of automating the information systems development and reducing the share of labor expenditures for writing the source code actively discussed in scientific and research works recently. This is evidenced by the emergence and development of Model-driven-architecture (MDA) [1]. It is worth noting that each author sees the problem through the prism of their own knowledge. Considerable imprint on the vision of the problem leave and subject area in which the researchers works. Thus, for example, experts in the field of requirements management [2,3] trying to automate the process of user requirements fixation through specialized tools and use them for generating of application source code. Specialists in the field of software design and development [4] concentrate their efforts on creating of universal abstract application model, and then realize information system through configuring and customization of this abstract model.

Analysis of these papers shows that common idea of authors consist in an attempt to capture the acquirements about specific domain in the form of some models, that underlie the static structure of a development platform.

Due to this realization created models have generalizing character and serve as a means of constructing models specific information system, i.e. they are situated at a higher level of abstraction - a meta-level. Therefore, they can be called metamodels. There is some examples of the meta-model's definition:

- Metamodel is a model that defines the language for expressing a model [5]
- Metamodel is a model of model [6]
- In MDSD, a metamodel is a “model of the modeling language.” [7]
- Metamodel describes the possible structure of models written in that language, i.e., the “constructs of the language and their relationships, as well as constraints and modeling rules” [8]

The concept of metamodeling can also be correlated with data storage technologies. This will open up new possibilities for design and development of information systems in general.

In this paper the meta-database – the tool of storing description of the subject domain at the different levels of detailing – from domain metamodel to user data are proposed. In the first section of this paper we describe features of development environments based on meta-database, in the second section – meta-database concept. The third and fourth sections of this article describe models of meta-database.

II. META-DATABASE BASED DEVELOPMENT PLATFORM

From the viewpoint of information systems development, the metamodel is primarily the knowledge ontology of the developers about particular domain with its activities and tasks. Also, the metamodel is a tool for describing information systems and data models.

Today the process of creating and using a development environment consists of the following steps:

1. Formation of the metamodel based on the acquired experience and knowledge. Metamodel, as a rule, is represented as a set of classes, and by this reasons it presents the static structure of the environment.
2. Static structure of classes is supplemented by components that will allow to work with the metamodel. Such components can be tools for interface specification, configuration metamodel, input/output data, etc.
3. Metamodel, supplemented by components, implemented into the finished form - development environment.
4. The metamodel is supplemented by description of specific business processes by means of the appropriate

components, configuration of this metamodel occurs. The complete product (information system) is formed in the result.

Visually, the process is as follows (fig. 1):

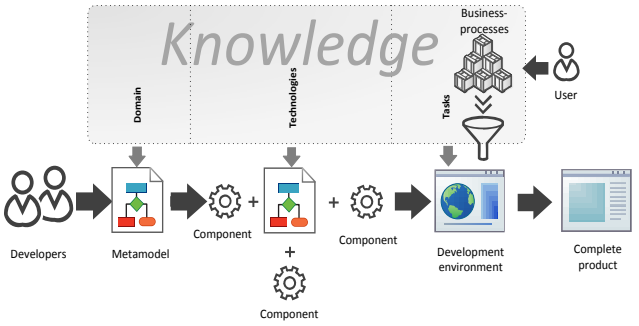


Figure 1. Metamodel-based development environment building process

Approaches that use a metamodel as a static structure of the development environment have capabilities to reducing the share of labor expenditures for writing the application source code. However, fig. 1 shows that the universality of these environments is limited by the range of problems from a particular subject domain, for example in the field of corporate web applications [4].

The problem of limitation this approach is fixing the metamodel as a static structure of a development environment. Creating more versatile metamodel - the task that is solved for today only in the development environment for object-oriented programming languages. Object-oriented paradigm allows to specify the elements of subject domain on abstract level, but it does not solve problem that is noted at the beginning of this paper - writing source code in such environments is the base component of the development process. Multiple approaches, such as code reuse and patterns are also oriented on source code, and it is not consistent with the direction of research outlined in the beginning of the paper.

In our opinion, the solution consists in empowering development platform by adding abilities to create new or modify existing therein metamodels. For this meta-model should be separated from the static structure of the development environment and is represented as object with complex structure stored in permanent memory (for example, like records in database), and the environment itself - have a means to describe and store metamodels (for example, a set of mechanisms for manipulating data and their structure). Such database, giving developers tools for describing metamodels of any structure, similarly to the metamodel can be called the *meta-database*. For the first time this term was formulated by Cheng Hsu et al. in 1990 [9], in those work were designated common meta-database properties, which is the integration into one the following aspects:

- knowledge about the company or the information system (i.e., operating, control, and decision);
- All models obtained at each stage of the development life cycle, ranging from multistage analysis to design and implementation;
- Data obtained in the process of using information system.

In our understanding, in terms of designated problem of automating the information systems developing process, *meta-database* is a tool that allows to represent a permanent memory descriptions of subject domains at various levels of detailing (the various meta-levels) - from metamodels to configuration of information system and user data.

In using the meta-database there could appears more abstract development tool - development platform. Development platform is a mechanism that reflects metamodels into the meta-database and thus it can allow to produce a less abstract development environments. These environments, in turn, give a complete product via concretization of metamodel to the level of current tasks (business processes) (fig. 2).

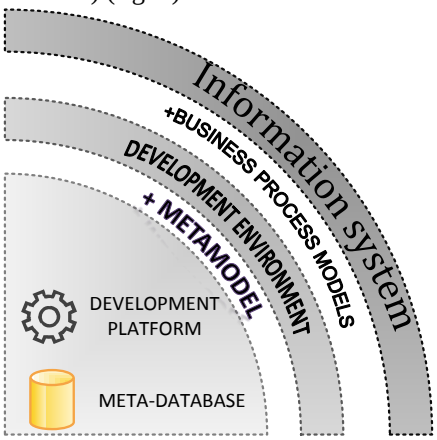


Figure 2. The concept of meta-database based development environment

Taking into account that the metamodel can be changed dynamically, without alteration the information system development platform, the process of building a development environment should look like this:

1. Existing meta-database based development platform that has a predefined set of basic components, if necessary, depending on the technologies knowledge complemented by missing components;
2. Domain experts (platform's users) placed metamodel into the development platform's meta-database, which is detailed to the level of selected implementation technologies.
3. The platform is configured to the level of the development environment which is oriented to concrete technology and a certain subject domain metamodel.
4. Users put the knowledge about current tasks in the form of business processes models into the development environment, and then they can try complete product (information system).

Visually, this process of creating a development environment may be shown as follows (fig. 3).

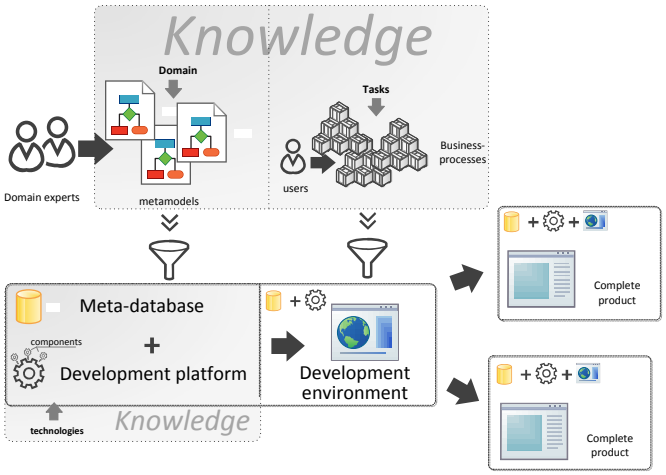


Figure 3. Using of meta-database to construct a development environment

Fig. 3 shows that the development platform can exist without any metamodel (in fact, the development platform in a static structure is described by the meta-metamodel, i.e., metamodel of higher level of abstraction). Knowledge of the developers from a particular subject domain, represented in the form of metamodel, are sort of configuration file for the platform. There can be several of such configuration files and in one platform we may have multiple development environments. In this way the development environment is constructed in accordance with the principles of meta-design and socio-technical environments [10,11].

This statement allows us to call the fig. 3 a meta-design model and to formulate the meta-database concept.

.

III. THE META-DATABASE CONCEPT

So, the meta-database is a tool that allows a unified strict way to represent in permanent memory knowledge about subject domains and results of operation their business processes. These two big aspects are divided into:

- metamodels of information system' classes for different range of tasks in any domain;
- logics of subject domain business processes;
- structure of the information system that is described in terms of the metamodel;
- user data collected as a result of business processes.

Thus unified representation may be entities (the objects with some structure information about which should be stored), attributes (characteristics of an entity) and relations (between entities), which may be a special kind of entity. Entities are relevant to the classes, entity instances - to the objects of classes, etc.

Such properties set up the number of requirements to the meta-database:

5. Existence of manipulating mechanisms not only for data but also for their structure. Since on the upper levels of detailing the domain structure is described primarily, and the data appears at the lowest level;
6. Standard format for querying the meta-database. Since the meta-database is integrated with a development platform, then regardless of its content should be standard ways to access data and their structure;
7. Independence from the structure and composition of stored information. As can be seen from fig. 3, the meta-database is constantly updated by some information; structure of whose can not be determined in advance that in the case of depending the meta-database structure on stored information will lead to malfunction of platform as a whole.

Implementation of these requirements depends on separation methods, which will be applied to divide storing instruments on the meta-levels. For example, in database technology, there is one meta-level - metadata. This meta-level strictly define the access path to the data and data structure. Often, metadata is a declarative tool that is supported on DBMS level. As a result, the metadata becomes static element of database and its changing leads to redesign software that uses data from database.

Meta-database should provide means of storing metadata, which can take an arbitrary structure, depending on the metamodel. By this reason, structure of the meta-levels described above is unusable - it has to be modified. To ensure the creation of meta-database the metadata, which is a tool for fixing the knowledge about data, should be descriptive and be contained inside the database (fig. 4).

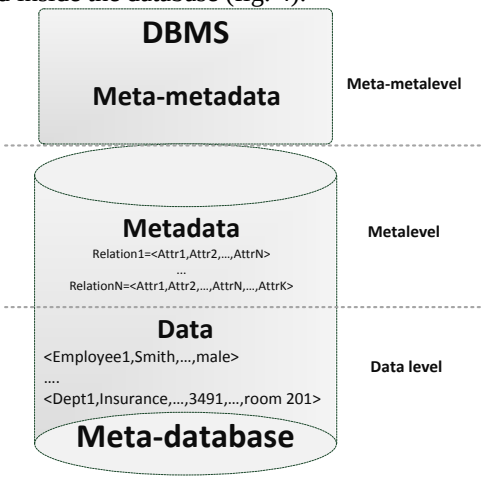


Figure 4. Meta-database concept

Fig. 4 shows that in process of using meta-database appear two meta-levels:

- Metalevel. Metadata - a means of representing knowledge about the database data. They are descriptive and are stored in the meta-database in a similar way with the data.
- Meta-metalevel. Meta-metadata - a means of describing the various metadata. It is a declarative tool that contains elements of the language for describing arbitrary metadata.

We can assert that the meta-database concept showed in fig. 3 is applicable for creating the information systems development platform since the metamodel, placed in the

permanent memory (in the meta-database), it is nothing else than the metadata, and configuration of metamodel reflecting particular application – the data in the database. In this case, there is a meta-metalevel – meta-metadata reflecting the meta-database structure and defining thereby a means of describing the various metamodels.

IV. THE META-DATABASE MODEL

Now turn to a more detailed presentation of the meta-database - its model. Here the key role is played by the strict representation in permanent memory of such aspects as domain knowledge and results of business processes operation. This strictness can be achieved by creating a formal meta-database representation model at the level of meta-metadata structure. Such formal representation will also allow to define accessing and manipulating methods for structure of stored metamodels and data.

As we told in second section of this paper, representation of any aspect from subject domain can be made with entities, attributes and relationships. Therefore, a formal tool for describing meta-database model may be of Codd's relational algebra [12], in which exist most similar concepts: entity - relation, attribute - attribute, connection - equal values of key attributes for two entities.

On the basis of the proposed concept, meta-database should combine in itself the lower level of representation - data of domain, and the first metalevel - metadata that store knowledge about data. When we create a formal model, this should be a fundamental requirement.

After analyzing several metamodels, described by the authors [4,13,14,15], we can define a unified set of their components:

8. Metamodel alphabet - entities reflecting the main aspects of information system produced by a specific metamodel. For example, use cases, data elements, reports, business logic, etc.
9. Entity attributes, adding of which leads to specification of metamodel.
10. Relationships between entities that define the structure of metamodel.
11. Hierarchy of entities, determining the order and direction of relationships.
12. Entity instances, reflecting the certain components of finished information systems.
13. Attribute values of entities by which metamodel is configured to the level of complete product.
14. Links between instances that represent structure of finished information system.

On the base of this classification meta-database model can be defined in terms of Codd's algebra by following system of relations:

$$M \equiv \langle E, A, S, L, R, V \rangle$$

where:

E – relation «Entities»

$$E = \{e_i\}$$

A – component «Attributes»

$$A = \{< a_i, t_j >\},\ t_i \in T$$

S – component «Entity structure»

$$S = (E, A, F), \pi_1(F) = E', E' \subseteq E,\ \pi_2(F) = A', A' \subseteq A$$

L – component «Relationship structure»

$$L = (E, A, F),\ \pi_1(F) = E', E' \subseteq E, \pi_2(F) = E'', E'' \subseteq E,\ \pi_3(F) = A', A' \subseteq A$$

R – component «Entity instances»

$$R = (E, I, F),\ \pi_1(F) = E', E' \subseteq E,\ \pi_2(F) = I$$

V – component «Attribute values»

$$V_{t_i} = (I, A, D, F),\ \pi_1(F) = I'', I'' \subseteq I,$$
$$\pi_2(F) = A', A' \subseteq A, \pi_3(F) = D'', D'' \subseteq D', D' \subseteq C_i$$

To prove the applicability of this model, consider the Ecore metamodel – the base for Eclipse Modelling Framework [14].

For example, we take two objects form Ecore: ENamedEement and ETypedElement, which is connected by hierarchical relationship without inherited characteristics (fig. 5).

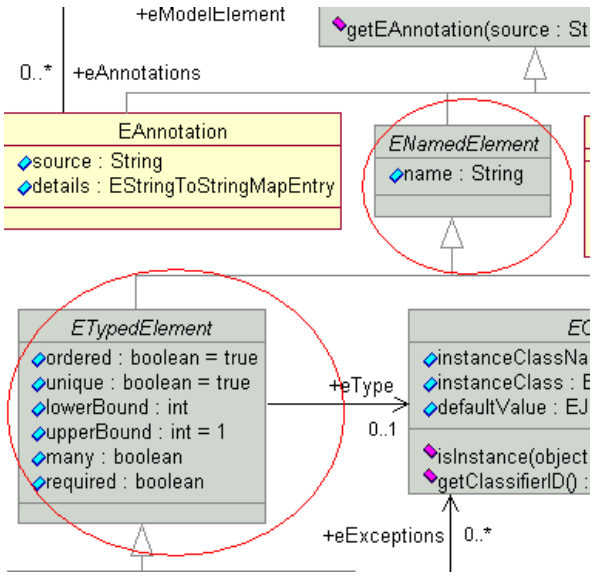


Figure 5. Ecore metamodel elements

Selecting entities:

E = { ENamedEement, ETypedElement}

Selecting the attributes and matching them with data types:

A = {<name,t3>,<ordered,t1>,<unique,t1>,<lowerBound,t2>, <upperBound,t2>,<many,t1>,<required,t1>}, где t1=’boolean’, t2=’integer’, t3=’string’

Matching the attributes and entities, defining the structure of entities:

S = {<ENamedEement,name>,<ETypedElement, ordered>,

<ETypedElement, unique>,<ETypedElement, lowerBound>, <ETypedElement, upperBound>,<ETypedElement, many>,

<ETypedElement, required>}.

Defining link structure:

L = {<ENamedEement, ETypedElement,0>}

Thereby, we described two objects of Ecore metamodel [14] by using of proposed formal model.

Let us suppose that in process of metamodel configuration appears certain instance of ENamedEement class and related to it instance of ETypedElement class. In our formal model this will be represented by next way:

Fixing of class instances:

R = {<ENamedEement,instanceID1>,<ETypedElement,instanceID2>}

Filling in values of attributes:

V = {<instanceID1, name, MyObject>,<instanceID2,ordered,true>,

<instanceID2,unique,true>,<instanceID2,lowerBound,-1>,

<instanceID2,upperBound,1><instanceID2,many,false>,<instanceID2,required,true>}

Fixing of relationship between class instances:

V = {<instanceID1, 0, instanceID2>}

Thereby, we showed creating of two class instances that appear in the process of metamodel configuring by using of proposed formal model. Such class instances could represent certain aspects of information systems or data. The given example and analysis of other known metamodels shows applicability of proposed formal model for its unified representation. More detailed representation of meta-database formal model and its utilization is described in [16].

V. THE META-DATABASE REALIZATION AND UTILIZATION

Guided by the fact that relations in Codd's algebra may be mapped in corresponding database tables and their attributes - in the fields of these tables, we implemented meta-database on the base of relational DBMS.

The principal difference between a simple relational database and meta-database is showed by three-dimensional representation in basis "entity-attribute-value" (fig. 6,7).

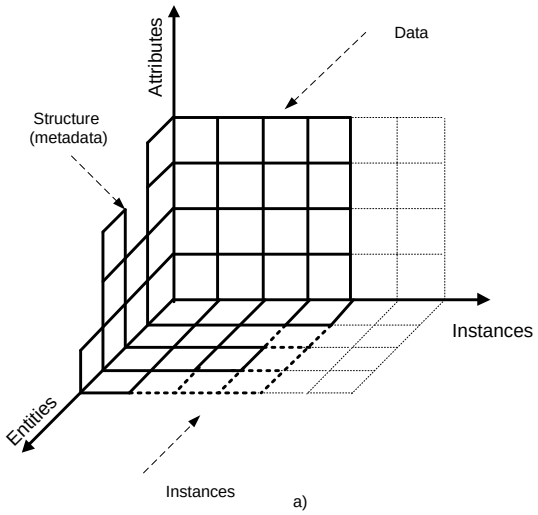


Figure 6. three-dimensional representation of a relational database in basis "entity-attribute-value"

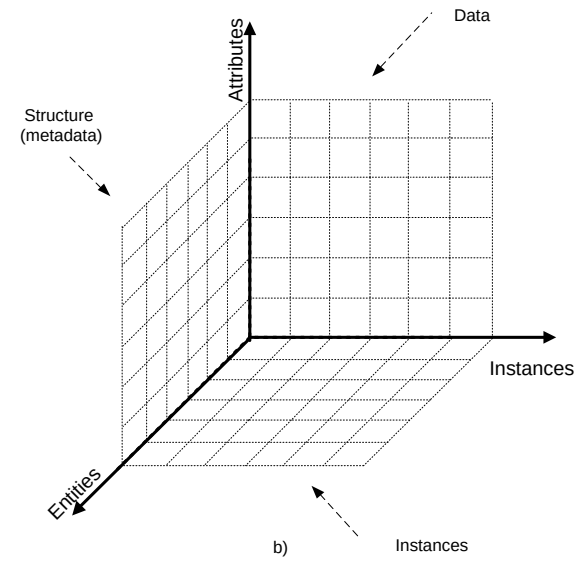


Figure 7. three-dimensional representation of a meta-database in basis "entity-attribute-value"

Fig. 7 shows that describing domain metamodel a-priori is not necessary for the meta-database – meta-database is ready for utilization even if it has no one entity and no one attribute. Using of a relational database (Figure 6) require pre-determine the structure of metadata and by this reason it is not suitable for resolving problem outlined in this paper.

We make some assumptions before giving the meta-metadata structure (structure of meta-database tables):

15. Metadata can be grouped and be represented by following corresponding tables:

- Hierarchical directory, which includes all metadata, barring "entity-entity" and "entity-attribute" relationships. Metadata stored in this directory is applicable for representing entities and attributes with the same efficiency, and therefore they may be represented by a single table.
- Attributes and relationships directory, which entries will refer to ascribable elements of hierarchical directory. Implemented as a separate table.
- Entity instances directory that refers to the elements of a hierarchical directory. Implemented as a separate table.
- Attribute values may be grouped by types into the similar tables for the purposes of efficiency. Ownership of these values will be defined by references to the metadata tables – name of field, which stores values, is not an attribute name. Attribute values make up the appropriate subschema of tables.

Whereas these assumptions, we represent the structure of meta-database as follows (fig. 8):

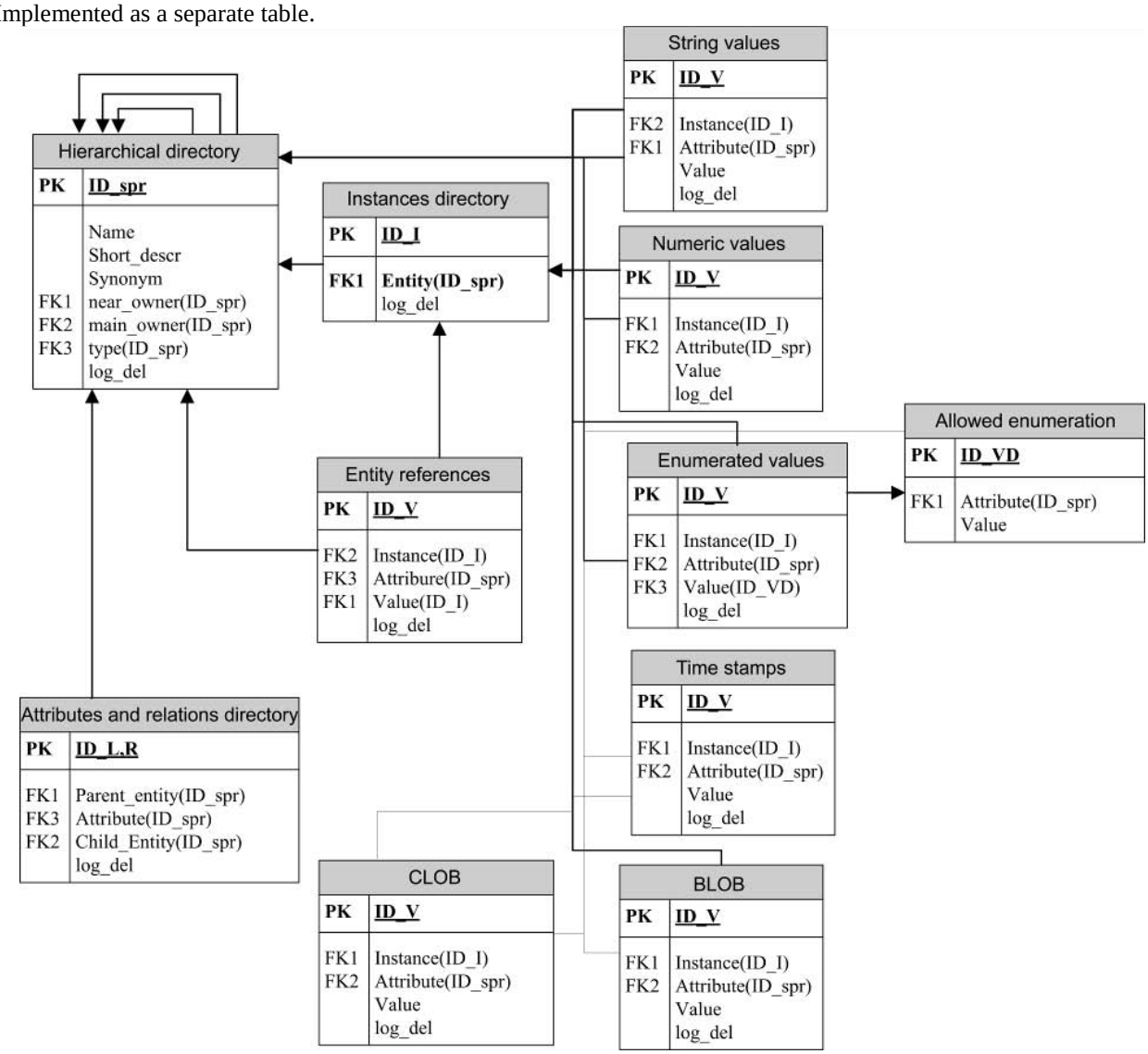


Figure 8. Logical model of meta-database

Such structure of data and metadata tables allows to implement all principles underlying the definition of meta-database, and it is satisfy the meta-database conception presented in fig. 4. In this case meta-metadata, which defining the rules of making metamodels, is the structure of meta-database logical model.

Summing up, we can formulate the following features meta-database:

- No need for a-priori description of stored data structure;
- Metadata is a descriptive. As a declarative tool we use meta-metadata;
- Independency of accessing data mechanisms from data structure.

Implementation of the meta-database is called SiDB (Structire-Independent Database) and described in detail in [17].

No less important question related to the meta-database is its utilization. For today the vast majority information systems are based on object-oriented paradigm. As a result, interact programs with relational databases is difficult - representation of application objects do not always correspond to representation of these objects in a relational database tables. To resolve these mismatch problems developers apply technology of object-relational mapping (Object-Relational Mapping). Actually, ORM is a layer between application and database that provides one mapping of data between two models (fig. 9).

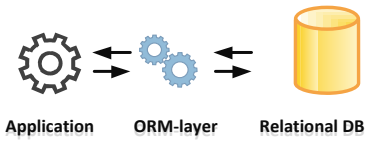


Figure 9. Using of ORM-layer

The main disadvantage of using ORM-layer is considered a loss of productivity. From this perspective, addition of a meta-level over the metadata would lead to even greater costs on the conversion and searching of data. However, using of metadata as a descriptive tool, which is not regulates strict access paths to data and permits access to attribute values directly in aggregate with a fixed structure of tables allows to replace ORM-layer by SQL-queries generator (fig. 10). By means of such generator any data from meta-database can be represented in the required form for particular application.

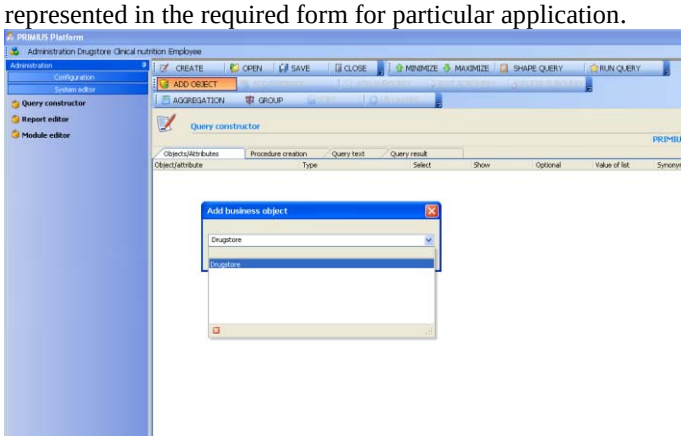


Figure 10. SQL-query generator interface

Thereby, building up another layer, that involved in the process of working with data does not occur (fig. 11).

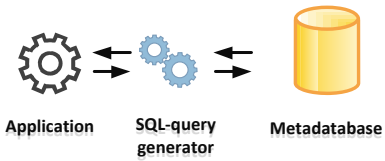


Figure 11. Scheme of interaction between the application and the meta-database

Lack of cost on data conversion, together with the possibility of using compiled stored procedures to access frequently requested data allow to achieve an enough level of performance, which is less than 10% inferior to relational databases created on the classical technology [18,19].

The effectiveness of the development platform is corroborated by its commercial use for automation of several social protection institutions in the Southern Federal District of Russia. This platform is called PRIMIUS (fig. 12) and in the working process is based on interpretation of the metamodel, which is created in terms of knowledge gained during many years of work in this area. The metamodel is implemented as entries in the metadata directories of SiDB.

Figure 12. Interface of development platform PRIMIUS

Information systems of any complexity is developing by means of platform «PRIMIUS», both for individual departments, where the volume of stored data is relatively small, and for organizations with a wide range of workstations and complex logical structure of stored data.

CONCLUSION

Proposed in this paper the meta-database is an effective solution, with close to a relational database performance measures. Meta-database has the following beneficial properties:

- storing of metamodel, information system model and user data simultaneously;
- absence of restrictions on the stored data structures;
- storing is not only the elements of models and relationships between them, but and the properties of these relationships;
- no need for a-priori description of the stored data structure;
- presence of means for formal description and data manipulation.

Due to these properties using of the meta-database allows to solve the next important problems:

16. Reducing the share of labor expenditures for writing the source code. Meta-database allows to build a development platform based on the paradigm of metamodeling, That allows the developers to generate source code by using of the models stored in meta-database.
17. Boundedness of development platforms which use metamodels as an aspect of the static structure. Built on the base of meta-database information systems development platform can contain a variety of metamodels, and permits their changing without loss of efficiency.

Current results of our work allow to form a basis for further research, among which the following main areas:

- Meta-database optimization and searching for its high-performance implementations;
- Creating tools for formalization description of subject domain and information system;
- Creating automate tools for the information systems development process;
- Creating tools for supporting variability of requirements and information systems.

REFERENCES

[1] Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. IEEE Software, 20(5):36–41, IEEE Computer Society, 2003.

[2] Carlos Rossi, Antonio Guevara, Manuel Enciso, José Luis Caro, Angel Mora. A Tool for user-guided database application development - Automatic Design of XML Models using CBD. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, Volume 2, p.195-201.

[3] Xufeng (Danny) Liang, Christian Kop, Athula Ginige, Heinrich C. Mayr: Turning concepts into reality - Bridging Requirement Engineering and Model-Driven Generation of Web Applications. ICSOFT 2007, Proceedings of the Second International Conference on Software and Data Technologies, Volume ISDM/EHST/DC p.109-116

[4] Athula Ginige. Meta-design paradigm based approach for iterative rapid development of enterprise web applications. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, Volume 2, p.337-343.

[5] Meta-Object Facility (MOF) standard. http://www.omg.org/mof/

[6] Gilles Dodinet, Michel Zam and Geneviève Jomier. Coevolutive meta-execution support - Towards a Design and Execution Continuum. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, Volume 2, p.143-151.

[7] Stephen J. Mellor, Kendall Scott, Axel Uhl, and Dirk Weise. MDA Distilled: Principles of Model-Driven Architecture. Object Technology Series. Addision-Wesley Longman, Amsterdam, The Netherlands, 2004.

[8] Thomas Stahl and Markus Völter. Model-Driven Software Development: Technology, Engineering, Management. Wiley & Sons, 1st edition, 2006.

[9] Hsu, C., Bouziane, M., Rattner, L. and Yee, L. "Information Resources Management in Heterogeneous, Distributed Environments: A Metadatabase Approach", IEEE Transactions on Software Engineering, Vol. SE-17, No. 6, June 1991, pp. 604-624.

[10] Fischer, G. (2007): "Designing Socio-Technical Environments in Support of Meta-Design and Social Creativity", Proceedings of the Conference on Computer Supported Collaborative Learning (CSCL '2007), Rutgers University, July pp. 1-10.

[11] Fischer, G., & Giaccardi, E. (2006) "Meta-Design: A Framework for the Future of End User Development." In H. Lieberman, F. Paternò, & V. Wulf (Eds.), End User Development — Empowering people to flexibly employ advanced information and communication technology, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 427-457.

[12] E. F. Codd: Relational Completeness of Data Base Sublanguages. In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987, San Jose, California : (1972)

[13] Gonzalez-Perez, C. and B. Henderson-Sellers, 2008. Metamodelling for Software Engineering. Chichester (UK): Wiley. 210 p

[14] “Ecore” http://www.eclipse.org/modeling/emf/?project=emf

[15] L. Lyadova. Technology for creating a dynamically adaptable information systems // Proceedings of International Conference on Artificial Intelligence and Systems (AIS’07). – Moscow: Fizmatlit, volume. 2, 2007.

[16] Y. Rogozov, A. Sviridov, S. Kucherov. An approach for formal representation of metamodels. Proceedings of first international conference on Actual problems of information systems and processes constructing. Taganrog: SFEDU, 2010, p. 9-15.

[17] Youri I. Rogozov, Alexander S. Sviridov, Sergey A. Kutcherov, Wladimir Bodrov. Purpose-driven approach for flexible structure-independent database design. Proceedings of the Fifth International Conference on Software and Data Technologies, ICSOFT 2010, Volume 1, p.356-362

[18] S. Kucherov. Performance evaluation of the statistical DB structure. Proceedings of seventh All-Russian conference «Information technologies, system analysis and management». - Taganrog: SFEDU, 2009. p. 138-141.

[19] Y. Rogozov, A. Sviridov, S. Kucherov. Performance evaluation of the structure-independent database SiDB for the purposes of full-text search. Proceedings of first international conference on Actual problems of information systems and processes constructing. Taganrog: SFEDU, 2010, p. 196-199.

The Problem of Creating Multi-Tenant Database Clusters*

Evgeny A. Boytsov
Valery A. Sokolov
The faculty of the computer science,
Yaroslavl State University,
Yaroslavl, Russia
{boytsovea, valery-sokolov}@yandex.ru

Abstract — **SaaS (Software as a Service) paradigm brings both lots of benefits to end users and lots of problems to software developers. One of such problems is an implementation of a data storage which is able to satisfy needs of clients of a service provider, at the same time providing easy application interface for software developers and great opportunities for administration and scaling. This paper provides a brief review of existing problems in the field of organizing cloud data storages that are based on the relational data model and proposes the concept of architecture of RDBMS cluster dedicated to serve multi-tenant cloud applications.**

Keywords - databases, SaaS, multi-tenancy, scalability.

I. Introduction

One of the most notable tendencies in the modern software development industry is the shift to Software as a Service (SaaS) paradigm. The main ideas of this approach are the following:

- An application is developed as a system of distributed services interacting with each other.
- All computing power and infrastructure needed for operating an application is supplied by a service provider.
- A fee for an application is taken on the basis of actual usage.

This approach is one of cloud computing delivery models. The main advantage of its usage for customers is that all expenditures for deploying infrastructure required for correct and stable operation of software suit are taken by a service provider. This fact should eliminate the need for a customer to have his own IT staff and purchase a new computer equipment with every new release of an application. Besides, this approach allows to completely solve the problem of software updating, because now it is done in a centralized manner by the software company itself, that means that all customers always use the most recent (i.e. the most safe and featured) version of the application.

However, the «jump into clouds» brings not only benefits, but also new problems mostly for developers and administrators of such systems.

It is known that most of enterprise-level applications are based on interaction with relational databases. Any customer relationship management system, docflow or enterprise accounting management system needs to store somewhere data used by the application . The de-facto standard for such data storages are RDBMS. In recent years there was a tendency to move most of the application logic to the database tier expressed in appearing procedural extensions of the SQL language, in which developers wrote stored procedures and packages of them. Modern RDBMS are able to process very large arrays of data, fulfill very complex data selection and data manipulation queries. The complexity of most powerful such systems is comparable with the complexity of modern operating systems. Some companies even create a specialized hardware for them. Most software development specialists are familiar with the SQL language and principles of data organization in RDBMS.

A typical scenario of RDBMS usage in traditional on-premise application is the following. A developer describes a required data structure (tables, indexes, views, constraints e.t.c.) and selects a database server which is the most appropriate for a «scale» of the task being solved and capability requirements (the choice here is very wide, from embedded libraries, like SQLite, to powerful enterprise-level distributed database clusters, like Oracle or DB2). After the choice has been done, a developer team defines an application logic based on interaction with the database server, and the application with the database server is installed at the customer's hardware. The application can be tuned in place to satisfy special needs of the customer. The most advanced applications are able to extend their data structure in place and to add an additional logic, usually using one of the script languages for such purposes. System administrators configure the user rights and requisites and upload necessary initial data to the database. After that the application is ready for use. In the context of the discussed problems there are two key moments of this process:

- Data and a database server are hosted by a customer, thus their safety and availability (except the case of data corruption by the application itself) are under responsibility of customer's IT-staff.
- Access rights are granted at the database level by configuring users and departments rights inside a company.

In the case of a cloud application, data are hosted by (and thus are under responsibility of) a service-provider which undertakes to provide instant and fast access to them for tens or

* This work is supported by the RFBR grant # 11-07-00549-a.

hundreds of thousands of its clients concurrently. Violation of any of these requirements (speed and availability) will cause penalties to the service provider and, that is much more important in the cloud industry, will worsen the image of the provider. A typical service level agreement for a cloud service guarantees its availability of 99%. These circumstances lead to the following conclusions.

- Maintenance of a cloud application implies large expenditures to organization of the data storage, caused by the need to store (most likely, in multiple instances to guarantee a required speed of access) data of hundreds of thousands of clients and their backup copies in order to restore in case of failure.
- Maintenance of a cloud application implies large expenditures on support of infrastructure, required to store data. This includes organization of monitoring resources availability, efficiency of their usage, backuping and restoring, organization of application data structure update and much more.

These considerations, in turn, lead to the following conclusion: an attempt to solve problems of scalability and management of a storage infrastructure using the existing technologies (which were used for designing and development of traditional «boxed » applications) would require very large expenditures, thus drastically limiting a barrier of entry to cloud business. That is why a common desire of SaaS vendors is to minimize costs of data storing and to find architectural solutions that would lead as much as possible to such a minimization without compromising performance and functionality of the application,.

One of such solutions is a multi-tenant application (thus also database) architecture. The main idea of this approach is to share one instance of the application among many tenants (companies subscribed to the service), thus drastically reducing expenditures to application servers, web-servers and associated infrastructural elements. An application design according to such architectural principles imposes some restrictions to functionality, but it brings unprecedented opportunities to scale the solution. Every instance of an application server lives «from request to request», without saving its state in internal data structures that allows, having sufficient physical (or virtual) computing power, to set up an unlimited amount of application instances to serve clients.

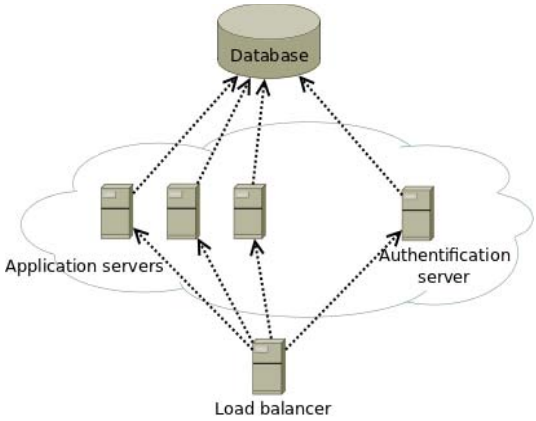


Figure 1. Simplified typical SaaS architecture

However, these considerations do not apply to database servers. Figure 1 shows a simplified architecture of a typical SaaS application, but even in this diagram we can see that a database server is the first candidate to become a bottleneck as the system grows. The reason for this lies in the fact that in contrast to application servers database servers scale poorly. To be more precise, application servers are able to scale well just because they descend most of load to the level of database servers, just often generating SQL queries and performing simple post-processing of the result. The database server should provide a reliable data storage, fast access, transactional integrity and others. A trend in recent years, when the most of the application logic moved to the database level, increased the load on this component of the system even more. In traditional systems the problem of scaling is less crucial, because a typical modern RDBMS in a single instance is able to serve needs of a medium company or even a fairly large one. Even if the problem of scale appears, it can be solved by formation of database clusters which consist of some powerful servers with a huge disk storage and database partitioning. However, this approach is not applicable to cloud solutions, because the total amount of data of all customers and the number of different queries, that they have to perform, are beyond the capabilities of any modern cluster.

II. Modern ways of organizing a multi-tenant architecture in cloud solutions

There is some experience in a field of organizing cloud data storages [1,2,3]. There are both completely new technologies based on absolutely new approaches and architectural solutions that allow to get multi-tenant environment in traditional RDBMSes.

A. *NoSQL databases*

The usage of NoSQL databases is one of the approaches that is widely used in the software development industry to organize a cloud data storage. This term denotes a wide range of technologies and solutions, the main characteristics of which is the emphasis on speed and ease of scalability at the expense of other functionality. The data in such databases are stored in separate records which are not limited by a predefined format, there are no terms like «NoSQL database normalization» or «NoSQL database data structure», in fact, these databases are just key-value storages. Such systems are simpler than traditional relational databases and provide a relatively small amount of data types and abilities to manage them. But the lack of features is compensated by very high performance and unlimited scalability. High performance is achieved by the simple structure and data indexing, scalability — by absence of complex data arrays and relations. Such databases can be used when a complex selection of large data arrays is not required and the application logic is based on the processing of individual objects or documents.

B. *Multi-tenancy in RDBMS environment*

In the case, when the application logic requires the complex processing of data, transactional integrity and other features that are currently unavailable in modern NoSQL databases, software developers have to return to the usage of traditional RDBMS. At the moment, there are two main approaches to designing multi-tenant relational database.

- Usage of shared tables for all clients with the attachment of a tenant identifier to every record — this is the shared table approach.
- Creation of the own set of tables for every tenant (usually, these tables are united into one database schema) — this is the shared process approach.

Both approaches have their pros and cons.

1) Shared table approach

This approach is the most radical one in answering a question of sharing server resources. The usage of this approach requires adding a special column to every table in the database which stores a tenant identifier for distinguishing data of different clients. Every SQL query to the database of such an architecture should be supplemented with an additional WHERE/HAVING predicate that leaves in the query result only records that belong to a specific client. There are also some projects of SQL extensions [4] that allow adding such predicates automatically, but at the moment these extensions are only concepts under development and research. The advantages of the shared table approach are the following:

- better usage of a disk space;
- small size of a data dictionary;
- better usage of a query planner's cache (i.e. shorter time of query analyzing and generation of its execution plan).

This approach has some disadvantages.

- The enlarging of the size of database tables and their indexes [5]. This drawback results in the requirement of very high qualification of developer of database queries, because any query for which database query planner is not able to generate an effective, indexes-based execution plan, will «hang» a database server for a large period of time. The most unpleasant thing in this situation is that usually the size of data of one tenant is relatively small and the query that operates on a data set with millions of records would return just a pair dozens of rows.
- The need to add the predicate of selection of a current tenant's data. This drawback leads to access errors, when users of one tenant can see data of another tenant. Such errors may be very destructive for the reputation of a service provider .The above [4] concepts of extensions of the SQL language are possibly able to solve this issue.
- The complexity of replication and backup copying of separate tenant's data. Since a tenant's data are «smeared» across the database, it is very hard to develop a generalized mechanism of their extraction and recovery.

In general, this approach shows good results, when application data schema is not large (there are not many tables) and a typical query is relatively simple (it does not contain joins of tens of tables, complex orderings and grouping, nested subqueries). If the above conditions are met, this approach to designing a database allows the most effective usage of a disk space and other hardware resources, storing data of tens of thousands of tenants in a shared database.

2) Shared process approach

This approach occupies an intermediate position in solving a problem of sharing server resources between complete isolation of a tenant's data in a separate database and a shared storage of them in the shared table approach. The separation of the tenant's data is achieved by creating its own set of database objects (tables, views, e.t.c.) for each tenant. The advantages of this approach are the following.

- Unification of the code of database queries and ease of writing new ones. In contrast to the shared table approach, queries are known to operate only the current tenant's data which usually have a relatively small size and therefore do not require a lot of memory and other database server resources for their execution [5].
- Relative ease of backup copying and replication of data of a single tenant. Because the tenant's data are grouped together in their own schema, they can be easily separated from others.
- Decrease of data security risks. It is much harder to make a mistake when writing a SQL query so that users of one tenant could be able to get access to data of another one.
- Simplification of system administration. Since a single tenant's data are a small logically separated database, they are human-readable and can be analyzed and corrected by standard database administration tools, if needed.

But there are also some drawbacks of this approach.

- Usage of this approach makes data dictionary of a database very large and heavyweight. Database metadata tables contain millions of records and are expensive to access. Every new database object makes them even larger.
- Because of a data dictionary enlargement, a query planner is unable to use its cache effectively, that makes him to generate a new plan of execution for almost every incoming query [4].
- A disk space is used less effectively than in the shared table approach [4].
- If a data structure change is required, it will take a very long time to complete, because of the large amount of database objects.

In general, this approach shows good results (to be more precise, there is no real alternative), if an application data structure is complex (contains a lot of objects) and a typical query selects data from a large set of tables, makes nested subqueries and other complex data manipulations.

III. Limitations of existing approaches and goals of the research

Despite the fact that they are not directly supported by most of database engines, both approaches are successfully

used by the software development industry. However, generated databases are very large and complex and therefore they are hard to manage. But every cloud application that aims to have a large user base has to operate on dozens of databases of such a complex structure. It is physically impossible to place all clients into one database. The highest level of database resource consolidation known today is about 20 000 tenants in one database with a relatively simple data structure. A simple calculation shows that even with such a high degree of resource consolidation, a company would require 50 database servers to serve 500 000 tenants, storing one backup copy of data for each of them for load balancing and data protection against failures and errors. In reality, such system would require much more database servers.

But the quantity of database servers is not the only problem in organization of a cloud cluster. Even more significant point is the load balancing for optimal usage of computing power and disk space at the entire cluster level. The nature of a cloud application is that the load on it is unpredictable, it may rise and fall like an avalanche and "burst" of activity can occur from a variety of tenants. The load may account for both the CPU and the network adapter of database servers, which occurs when the activity of tenant users increases, and on its disk drives, which will inevitably occur with the data accumulation. To provide the required level of service, an application should be able to dynamically adapt itself to changing conditions by automatically redistributing available resources. At the moment, there are no software systems that are able to solve this problem, as there are no clear requirements and approved algorithms for them. This research has the following goals:

- Development of algorithms of load balancing for multi-tenant cloud database clusters.
- Research of developed algorithms for correctness and safety, including imitation modelling and stress testing.
- Development of complex solution for organizing multi-tenant cloud database clusters using ordinary servers.

IV. Solution requirements

The developed system will be intended for using by small and medium-sized software companies, and therefore it should be designed to meet their needs and capabilities. The following points can be noted:

- Reliability and maximum guarantee of data safety. The reputation is extremely important for a cloud service provider, because his clients trust him their data and for many of them this is a rather difficult decision. This decision will become much more difficult to make, if there are clients' data corruption or loss.
- Efficiency. A multi-tenant cloud cluster management system should use available resources in the most efficient way, providing maximum performance of an application.
- The similarity to traditional DBMS with minimal possible corrections for the cloud application specifics. It is known that most software developers have some experience with traditional RDBMS and the SQL language and know main principles of the relational data model. This is the significant benefit that should be used.
- Horizontal scalability. Horizontal scalability is the ability of a system to increase its performance by adding new servers to existing ones. This differs the horizontal scalability from the vertical scalability, when performance of the system is increased by upgrading the existing servers. The horizontal scalability is preferred to vertical, because it is cheaper and potentially allows to infinitely increase the performance of the system.
- Ease of administration. If a service provider aims to serve lots of clients, it has to deal with a very large and complex infrastructure and its manual administration can lead to management chaos and system unmaintainability. To avoid this, the cluster management system should provide maximum automatization and tools for real-time system monitoring.

Let us list the main characteristics of multi-tenant databases for cloud applications, which should be taken into account, when designing a cluster management system:

- The huge aggregate size of stored data. As the provider is to serve dozens and hundreds of thousands of clients, the total amount of data which it is responsible for is huge and constantly growing with an unpredictable speed.
- The small size of a single client data. Since we consider multi-tenant solutions, this solution is likely to aim at dealing with small and medium-sized companies (so called «Long Tail» [1]), and therefore the number of users and the size of data of an average tenant is not large.
- The presence of shared data. Usually any cloud application has some set of data which is shared among all tenants of the provider. Such data include various directories and classifiers. Accessing them is almost always read-only. Such data should be stored in a single copy, providing their replication to all database servers in a cluster.
- The need for data backup.
- The need for data replication. Like in traditional DBMS, data replication is used to balance the load of database servers. The distinction is that often the replication in cloud solutions is partial, i.e. only the data of one or some tenants are replicated.

Based on these requirements and features, we present the proposed project of the cloud cluster management system.

V. The architecture of multi-tenant database cluster management system

When designing the cluster management system, it is important to think about the comfort of its users in advance. There are two categories of users of the system: application developers and system administrators. The first category deals

with the system API to create their applications based on multi-tenant cluster technologies. The ideal interface for this category of users should ease the interaction with the system and hide cluster implementation details behind an additional layer of abstraction. From the point of view of a software developer, the cluster should be considered as a set of independent databases for each tenant with a single entry point (i.e. database connection string). A desired application interface should be the following:

```
Connect( params );

Execute( ClientID, «SELECT * FROM T1» );

Disconnect();
```

As for the second category of users (system administrators), the most important characteristics of the system are the ease of configuration and monitoring. Ideally, the configuration of the system should be as simple as possible and should be limited to specifying the physical structure of a cluster. Monitoring tools should provide reports that show the current state of the system in maximum detail.

The main idea of the proposed solution is to add a new layer of abstraction between application and database servers, functions of which are:

- The routing queries from an application server to an appropriate database server by the tenant identifier.
- Management of tenant data distribution among database servers, dynamic data redistribution according to an average load of servers and characteristics of different tenants activity in time.
- Management of data replication between database servers in the cluster.
- Management of data backuping.
- Providing a fault tolerance in the case of failure of one or some cluster databases.
- Analysis of resource usage and system diagnostics.

The system should be implemented as a set of interconnected services using its own database to support a map of the cluster and collect statistics on the system usage by tenants and characteristics of the load. The shared process approach is going to be used for tenants data separation at the level of a single database. The choice of this approach is explained by the fact that the system must be sufficiently general and have no knowledge about the data structure required for an application beforehand. Because one of the requirements of the shared table approach is to add a service column to every table in the database, it assumes much closer familiarity with the application data structure and thus its usage is difficult for the generic system. Moreover, the usage of the shared table approach requires very good query optimization skills and thus does not hide the underlying structure of the cluster from the developer. The general architecture of the proposed system is shown in Figure 2.

We proceed to a more detailed consideration of the above-mentioned functions of the system. The proposed solution assumes the appearance of a new element in a chain of interaction between application and database servers. This new

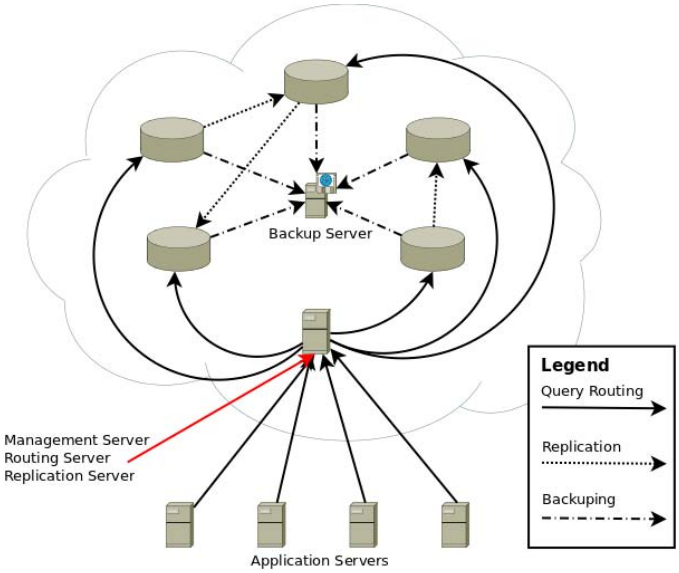


Figure 2. Multi-tenant database cluster architecture

element is a dedicated server which transparently for application servers routes their queries to an appropriate database server, basing on a tenant identifier, provided with a query, characteristics of the query and statistics of the current system load. This is the component application developers will deal with. In fact, this component of the system is just a kind of a proxy server which hides the details of the cluster structure and whose main purpose is to find as fast as possible an executor for a query and route the query to him. It makes a decision basing on the cluster map.

It is important to note that the query routing server has a small choice of executors for each query. If the query implies data modification, there is no alternative than to route it to the master database for the tenant, because only there data modification is permitted. If the query is read-only, it also can be routed to a slave server, but in the general case there would be just one or two slaves for a given master, so even in this case the choice is very limited.

Besides, it is important to mention that the discussed component of the system can not use expensive load balancing algorithms, because it must operate in real-time. All it can use is its own statistics on the number of queries sent to a specific database server of a cluster, the execution of which has not yet been completed. Basing on this runtime data, it must decide where to send the next query.

The implementation of this component should give a significant benefit in performance and ease of cluster administration.

The second component of the system is the replication and backup management server. Its functions are clear from the name but it is important to note that, unlike the traditional databases, in multi-tenant solutions the replication is almost always partial, i.e. only some part of data is replicated. For example, the data from the first tenant schema can be replicated to one database, from the second tenant schema — to another and the third tenant schema itself can be a replica of a part of the second database from a cluster. Once again we recall that the data change request can only be executed by the master database of the tenant and this consideration should be taken

into account during the distribution of tenant data among servers to avoid hot spots.

The third component of the system, its peculiar "circulatory system", is a set of agent-services placed at the same machines as database servers. These small programs-daemons are to collect statistics about the load of the server (usage of CPU, RAM, network adapters and disk space) and monitor server state in the case of failure. All the information collected is sent to a central server for processing and analysing and will be used as an input data for the load balancing algorithm.

The last and the most important and complicated component of the system is the data distribution and the load balancing server. Its main functions are:

- initial distribution of tenants data among servers of the cluster during the system deployment or addition of new servers or tenants;
- collecting the statistics about the system usage by different tenants and their users;
- analyzing the load on the cluster, the generation of management reports;
- management of tenant data distribution based on the collected statistics, including the creation of additional data copies and moving data to other server;
- diagnosis of the system for the need of adding new computing nodes and storage devices;
- managing the replication server.

This component of the system is of the highest value, since the performance of an application depends on the success of its work. The key indicators that can be used to evaluate its effectiveness are:

- the average response time of a service (an average time between the arrival of a request and receiving a response to it);
- availability of a service (what percent of requests from the total number was successfully executed, what percent failed to meet a time limit or other parameters and what percent is not executed at all);
- the average load of database servers (whether servers are equally loaded, whether there are idling servers when others fail to serve all requests).

Obviously, the last criterion affects the previous two.

An algorithm of cluster load analysis and need for data redistribution should become the core of the load balancing system. This algorithm should make its decision about data redistribution, taking into account the following considerations.

- Performance of cluster servers. If the system is not homogenous, the proportions of its parts should be taken into account.
- Free resources available. If the system has free resources in its disposal, it makes sense to use them by creating additional copies of tenant data to increase the performance of the application. However, if the number of tenants begins to grow, the created redundant copies should be removed.
- The history of the individual tenant activity. If users of the tenant A actively use the application and users of the tenant B do not, it makes sense to move the data of the tenant B to a more busy server and create less copies of them, since they are unlikely to cause problems for the service.
- Preventing the creation of hot spots on writing the data in a context of replication organization. The system should distribute master servers for all tenants in an appropriate way, taking into account the history of the tenant activity.

Such an algorithm can be based on a variety of strategies, and currently it is not clear, which of them should be preferred. From this it follows that the most reasonable solution would be to implement several variants of the algorithm as a pluggable modules and choose the best one according to the results of imitation modelling and stress testing. It is very likely that such a version will not be found and the final implementation of the system will contain several versions of the algorithm which showed themselves as the best ones under some external conditions. In this case the choice of an appropriate version of the algorithm will be the task of a cluster management system administrator.

[1] F. Chong, G. Carraro, “Architecture Strategies for Catching the Long Tail“, Microsoft Corp. Website, 2006.

[2] F. Chong, G. Carraro, R Wolter, “Multi-Tenant Data Architecture“, Microsoft Corp. Website, 2006.

[3] K.S. Candan, W. Li, T. Phan, M. Zhou, "Frontiers in Information and Software as Services", in Proc. ICDE, 2009, pages 1761-1768.

[4] Oliver Schiller, Benjamin Schiller, Andreas Brodt, Bernhard Mitschang, “Native Support of Multi-tenancy in RDBMS for Software as a Service“, Proceedings of the 14th International Conference on Extending Database Technology EDBT '11 2011

[5] D. Jacobs, S. Aulbach, “Ruminations on Multi-Tenant Databases“, In Proc. of BTW Conf., pages 514–521, 2007.

Automation of QA in the project of DB migration from SQL Server into Oracle

Iakov Kirilenko & Eduard Baranov
Software Engineering Department,
Mathematics and Mechanics Faculty,
Saint-Petersburg State University,
Saint-Petersburg, Russia

Abstract— **This paper describes an automatic QA organization experience in the industrial project of DB migration from MS SQL Server 2005 to Oracle 11gR2. The resulting DB of the project is supposed to contain the same data and to have a functional correspondence with the initial one. The initial DB is quite huge: 6 terabytes of data and 2500 KSLOC of stored procedures. The documentation for the initial base is incomplete and outdated and doesn't correspond with the database in question. Functional specifications for stored procedures are missing, as well as tests. This article contains the description of the main problems solved during the project, solutions and an estimation of their applicability based on implementation experience.**

Keywords - database, data migration, testing, reengineering, quality assurance

I. INTRODUCTION

The authors are contracted to migrate the industrial database from MSSQL Server 2005 to Oracle 11gR2. There are main requirements to the project: all data must be migrated from the initial base and all functionality must be preserved. Also there is an afunctional and difficultly formalizable requirement to minimize changes in schema of the database and in signatures of stored procedures. This requirement was introduced by the customer to decrease the cost of the following adaptation for the new DB of client applications.

The project is organized as simultaneous progress in two ways: analytics of the initial DB and development of the set of tools for automatic migration. During the project DB migration process improves continuously to entirely automatic migration. For this purpose a set of tools is developed for automation of all predesigned steps: database schema transformation, migration of stored procedures and data transfer. At the same time a complex system for quality assessment is being developed.

Absence of the documentation, which describes functional behavior, was decided to be compensated by an automation of comparison between behaviors of the initial DB and the migrated one in functional scenarios that correlate with business use-cases. Functional complexity of the system (2.5 million lines of stored code) and huge amount of data (6 Tb) considerably complicates organization of the QA process. Additional changes made in schema and in stored code during migration also complicate the automation of the comparison.

II. RELATED WORKS

Methodologies for database migrations are described in several papers. In the article [2] the example of data migration methodology is presented. Migration process between DB with different data models, its risks and problems are described in [3] and [4]. Methodologies for legacy system migration are presented in [5].

Database migration projects have special risks and problems. Authors of [6] propose the data migration triangle for project management in this area. One of dimensions addresses quality assurance. Typical risks, testing and QA techniques are described in [7]. Testing during a database migration lifecycle is considered in [8].

Differential testing was initially proposed by McKeeman [9]. It is a special case of random testing to detect differences between different implementations. Regression testing is discussed in [10] where differential unit-tests are proposed for detecting differences between versions of the same unit. In paper [11] a tool which can identify the cause of regressions by trace analyzing is discussed.

There are a lot of works about data validation. The paper [12] describes it with emphasis on automation, quality and security of data validation process.

An experience of the migration testing can be found at [1].

III. QA PURPOSES

The main purpose of the migration project is to result in a new DB, which contains the same data and has functional correspondence with the initial one. Testing, especially the functional one, is necessary for inspection of migrated DB.

The initial DB is accepted as a model for the migrated DB by formulation of the problem. Behavior of the initial DB is considered as correct and the migrated DB must functionally correspond the initial one within the accuracy of documented changes, which contain renames, changes in a

database schema and consistent changes in procedures semantics.

Goals of the functional testing are:

- Verification of data migration completeness
- Functional correspondence between the initial DB and the migrated one.

Migration tool developing process also needs a control and permanent verification is needed for results of its work. So, objects for testing are:

- Code of the migration tool
- Data migration procedure
- Code for RDBMS Oracle

Functional testing must solve the following problems:

- To provide constant control for correspondence between generated code Oracle PL/SQL and projections designed in the migration tool
- To provide constant control for regression during the migration tool development
- To provide control for functional correspondence between initial (SQL Server) and migrated (Oracle) DBs
- To provide control for data migration correctness
- To provide everyday control for code migration completeness

The QA is executed in two main directions: generated code testing and the data migration process testing.

IV. MIGRATED CODE TESTING

A. Testing of Migratred Code Functionality

Functional testing process is based on a synchronous playing of prepared traces in two DBs (initial and migrated). Trace is a sequence of queries to the DB, which formalizes an interaction between the DB and client applications. The main requirement to the set of traces is sufficient functional coverage. It is a black-box technique, when only external effects are checked. They include output parameters, result-sets, changes inside DB etc.

First traces were created from testing scenarios which were given by the consumer. Traces were collected from industrial servers in order to get more precise information about functionality used in the maintenance. This helps to enlarge the trace set with more priority scenarios which cover more important functionality. Additional synthetic traces were also developed for testing of rarely used functionality.

Using only this kind of testing isn't convenient for the specific project. The whole migration process lasts several hours and it's too long to wait for results of small changes. So some kinds of errors (e.g. incorrect construction transformation) ought to be detected at earlier stages.

B. Early Determination of Defects

Primary migration tool testing is based on small tests, which cover main functionality. These tests are designed for early regress determination, so among them there are examples for all code constructions. A test set is executed automatically after every commit in a version control system and allows prompt detection of an incorrect construction transformation. Analytics and developers increase number of tests during the development of the tool.

In addition, automatic loading and compilation in Oracle are executed every day on procedures translated with the most recent version of the migration tool. They allow to control a number of correctly (syntactic correctness) translated procedures, and show errors appeared in code. This basic testing is especially urgent during the active development of the migration tool.

Testing based on procedures compilation isn't enough for providing syntactic correctness of the code in the specific project. A lot of procedures which contain critically significant functionality use dynamically generated queries. Additional functionality was developed for dynamic SQL, which allows a detection of statically (without procedure execution) lines of literals, which generate incorrect dynamic query for sure.

C. Testing by Trace Playing

The trace recording method is based on MS SQL Server 2005 embedded tools. Queries are captured and saved during the using of DB by customers. Traces for functional testing are recorded on the initial DB in a single-user mode. Each tester has an individual virtual machine. Virtual machine state is saved with deployed DB before trace recording. After recording traces are converted into a unified view which is based on XML. The unified view represents original structure of the initial trace. It consists of batches splitted in queries. It keeps an original text of each query, which was executed on SQL Server, and a text for Oracle. Queries for Oracle are generated automatically by the trace transformation with the rules which were used in stored procedures code transformation. For this purpose integration with trace transformer was added.

The trace playing always starts from the same saved state, determined by the saved state of the initial DB. The migrated DB is a result of migration process application to the saved state of the initial DB. So the trace playing always starts from two DBs in equivalent states. A synchronous playing is also executed in a single-user mode. It provides determined order of the stored procedures execution and a regular repeatability of the trace playing result. Distortions are observed only in results returned by disordered samplings, but the trace playing tool takes this problem into account during the result analysis.

Traces are played with a special developed utility. Result sets received on each step are being compared as difference sets A-B and B-A. If both of difference sets are empty then the result is accepted and translated code is considered to be functionally equivalent to the initial T-SQL one. Otherwise, an attempt to compare sets A-B and B-A is performed by rows and after that by columns for the error localization. Results of comparing are logged into a report. Each trace has

its own record. Errors which were found during the code execution are reported too.

For the QA improvement another monitoring was added which strengthens control on the equivalence of initial and migrated DBs. It looks for changes made during the trace playing inside DBs. For this purpose triggers on events INSERT/UPDATE/DELETE were added for every table in initial and migrated DBs. Triggers write down information about all changes made during the trace playing into a special audit table.

For each section of the trace triggers write down information about the fact of execution an operation on data. It contains a table name, an operation type (INSERT/UPDATE/DELETE), a number of changed values and a hash key of the value collection. This functionality is implemented in one trigger for all three operations on each table. After the trace playing values in the correspondent audit tables are compared. If any difference is found, correspondent tables are also compared. The result of using of this method shows that this testing can find differences in the number of deleted rows which can't be found by comparison of the returning record sets. Noticeable efficiency decreasing wasn't observed after the triggers addition.

D. Control and Providing Testing Coverage Completeness

Testing scenarios which were used for the trace recording were given by the customer. It is supposed that they cover sufficient functionality of the system. For test coverage (completeness) quality assessment source codes of the stored procedures were automatically changed on test servers. Logging instructions were added which allow to backtrace a sequence of operators in order they were executed with an acceptable accuracy. At the beginning of each line code section a command is added which inserts information about passing through a checkpoint into a special table. During the trace playing on the test server a log table is generated. With knowledge about all checkpoints and their places it's possible to count code test coverage using this log. Synthetic tests are counted separately.

For the testing coverage control traces from industrial server are also used, and it helps to determine how the test set covers wide used functionality. But the customer gave only a few industrial traces.

The most completeness cover could be provided with traces from the industrial server, collected during a long period of time. But there are some problems with their reproduction. Firstly, traces and the DB image must be consistently depersonalized before their transfer to third persons. It makes no difficulties to implement such functionality while having such analysis level of the migration tool, but it's impossible because of the project time limit. Another problem is the recording of the reproduced traces even with solved problem of the consistent depersonalization. Values returned in samples to client applications can depend on current state of the DB, for example IDENTITY column. In certain cases results made during parallel sessions are not comparable. Possible solution of the problem is to substitute IDENTITY generation (and a column type converter) in the initial DB for more suitable functions. But this substitution mustn't have influence on efficiency.

V. DATA MIGRATION TESTING

A DB size is impressive, so there is no guarantee, that functional tests can find an imperfection of data loading. So the QA of data loading procedure implementation is based on return codes, logs analysis of all system utilities used in a load chain, a data integrity control embedded in the DB and an additional control of the data loading result – a validation[2].

The validation makes it possible to assure that all data reloaded successfully to the new DB after all necessary documented transformations. This DB is a key component for a valuable part of customer business, so the validation is very urgent after the DB migration of such size. The database schema contains more than 2000 tables and at almost 10000 columns, some tables contain tens of millions of records. Foreign keys as an instrument of data integrity practically aren't used.

The validation process must check not only objects content but the whole database schema verifying existence and state of objects. Full validation for checking DBs equivalence with such volume DBs needs a huge amount of resources, especially a time resource. For a regular process another method is needed. It must require much less resources but have a good result confidence.

At first, validation by row counting was used for primary testing. Measure of success was a table's row number coincidence in initial and migrated DBs. Implementation was pretty simple. It was necessary only to count a number of rows in all tables and to compare results. Script was automatically generated during the database schema and stored code transformation. One of method's advantages is its speed. It has high speed, especially on tables which have primary or unique keys. On this tables number of rows is counted by index and full scan of the table is not performed. Even such simple method revealed unsuccessfully migrated objects. But this method doesn't verify objects values and this is a big disadvantage, especially in migration which is accompanied by the data type transformation.

For the validation result reliability improvement another method was implemented. It is based on hash keys comparison. In the initial DB hash keys are calculated for all columns in every table and results are saved in a separate table. Procedure for hash keys calculating have the following requirement: values of hash keys must be independent from table strings traversal order, because the rows order in the sample can be different and ordering is a very slow operation, it can strongly reduce the speed of the validation. At the current implementation the XOR operation is used. It is commutative and it is embedded into SQL. However research is being made in order to find another function which makes more qualitative hashing but at the same time is very fast. Calculation code is implemented in T-SQL. This code and table with hash keys are being migrated with database schema and stored procedures. Migrated hash calculation procedure on migrated data must give the same

value as an initial one on the initial DB. This also makes an additional testing of the migration tool.

VI. Results

Described testing strategy was implemented within the project. The unit test set contains a hundred of tests for different input language constructs. Permanent control over transformations of the constructions has helped to save a vast amount of men-hours. Compilation in Oracle has often showed a regress made in the previous day, so it wasn't difficult to isolate faults. Both controls have taken just several minutes which is nothing in comparison with migration process.

Trace comparison has been conducted with more than 400 traces and is still growing. Trace playing process takes 6 hours. As a result of the discovered differences investigation a lot of problems were found, and some of them forced to improve or change introduced projections. Moreover, this testing methodology has discovered problems in the initial DB (e.g. some queries return first element from unstable unordered selection, so the result can't be assured).

Evaluation of the testing coverage showed that testing scenarios which were given by the customer had covered about 14% of operators. Additional synthetic scenarios made it possible to increase this value to 33%. During the functional testing about 49% of procedures were executed. Moreover, a huge amount of dead code (more than 40% of operators) was found in the initial database, so the resulting coverage is enough. 20% were confirmed by the customer as acceptable test coverage. This number is based on previous experience in reengineering and correlates with Pareto's principle.

Data validation process has had two implementations. First implementation (by line counting) was fast, and it had found several losses during the data migration at early stages of the project. Next implementation based on hash keys comparison has helped to improve data migration process and it can provide rather high probability of the data migration correctness. Validation process has taken 6 hours on test servers which is much less than full validation.

VII. Conclusion

This paper presents an experience of the QA organization in a technically complex project of DB migration. Described methods were implemented and tested in practice and show their efficiency.

In spite of positive results of the current QA organization and automation some methods can be improved. The main direction of methodology improvement is supposed to implement trace recording and playing from the industrial server. In order to achieve this synchronization problem and depersonalization problem are needed to be solved.

References

[1] Sneed H.M.,"Selective Regression Testing of a Host to DotNet Migration", Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference J. ds, 1892, pp.68–73.

[2] Hudicka J., "The Complete Data Migration Methodology", Dulcian Inc., June 2000

[3] Chang-Yang Lin, "Migrating to Relational Systems: Problems, Methods, and Strategies", Contemporary Management Research, Pages 369-380, Vol. 4, No. 4, December 2008

[4] Maatuk A., "Migrating Relational Databases into Object-Based and XML Databases", Doctoral thesis, Northumbria University, 2009

[5] Wu B., Lawless D., Bisbal J., Grimson J., Wade V., O'Sullivan D., Richardson R., "Legacy System Migration : A Legacy Data Migration Engine", Proccedings of the 17th International Database Conference, October, 1997. pp 129-138

[6] Klaus Haller, "Data Migration Project Management and Standard Software – Experiences in Avaloq Implementation Projects", Proceedings of the DW2008 Conference, St. Gallen, Switzerland, 2008

[7] Matthes F., Schulz C., Haller K., "Testing & quality assurance in data migration projects", Software Maintenance (ICSM), 2011 27th IEEE International Conference.

[8] Patil S., Royy S., Augustinez J., Redlichx A., Lodha S., Vin H., Deshpande A., Gharote M., Mehrotrak A., "Minimizing Testing Overheads in Database Migration Lifecycle", The 16th International Conference on Management of Data (COMAD), 2010

[9] McKeeman W., "Differential testing for software", Digital Technical Journal, 1998

[10] Elbaum S., Chin H., Dwyer M., Dokulil J., "Carving differential unit test cases from system test cases", FSE'06, 2006.

[11] Hoffman K., Eugster P., Jagannathan S., "Semantics-aware trace analysis", PLDI'09, 2009

[12] Manjunath T., Ravindra S.,Mohan H., "Automated Data Validation for Data Migration Security", International Journal of Computer Applications (0975 – 8887)Volume 30– No.6, September 2011

[13] Microsoft Developer Network Library, http://msdn.microsoft.com/en-us/library/default.aspx

[14] Oracle Documentation, http://www.oracle.com/pls/db112/homepage

One approach to document semantic indexing based on multi-agent paradigm

George Sokolov
Computer Science Department
Perm State University
Perm, Russia
sokolovgeorge@gmail.com

Viacheslav Lanin
Department of Business Information Technologies
National Research University Higher School of Economics
Perm, Russia
lanin@perm.ru

The problem of search pertinence increasing with a low time-complexity is one of the major research issues in Computer Science. Semantic search as an alternative solution to this problem has a high time complexity. This paper describes the use of agent-based approach to reduce the time complexity of constructing semantic indexes used for searching.

Semantic indexing; agent; ontology; document

I. INTRODUCTION

Nowadays the information retrieval (from the Internet and off-line sources) is one of the major research areas in Computer Science. The main criteria of a successful search are the high relevance of search query information and fast response time. Traditional search engines typically use an approach «Bag of words» based on statistical methods to search for information. This approach takes precedence over semantic search methods is due to low time-complexity, low implementation complexity and satisfactory degree of relevance. One of the main areas of modern researches in the information retrieval is an increasing of search pertinence with a low time-complexity.

In syntactic search some indexes are built to find quickly the information required on some key words. By analogy let's introduce a concept of a semantic index. In this paper the semantic index is one-one correspondence between elements of the text and concepts from some ontological resource. There are different formats of the semantic indexes. Some of them are primitive (such as microformats hCard, Geo, microdata html5) and other formats are advanced (such as RDF, OWL, underlying the Semantic Web). In the semantic indexing there are two directions: the construction of semantic indexes and search for information on a semantic index. In this paper we will consider the construction of the semantic index (or the so-called semantic markup) for electronic documents.

The main problems of constructing semantic indexes are

1) high time-complexity (is due to various kinds of ambiguity that require paying respect of a context),

2) the problem of choosing ontology, which would be sufficiently complete to satisfy all search queries in an electronic document,

3) large amount of constructed semantic indexes and the problem of storage.

In this paper, the authors offer one approach of solving the first problem (the problem of time-complexity). Obviously, increase in the rate of the semantic indexing operation is required not one but several calculators, i.e. the parallelization of this operation is needed. The execution of the semantic markup operation requires the coordination of actions to resolve ambiguities. That's why simple asynchronous calculators aren't capable to solve the problem. According to the authors the most appropriate solution is using agent-based approach.

II. EXISTING APPROACHES

Solution to the agent-based semantic indexing problem can be obtained in two ways:

1) using of generic agent-based platforms that can decide a wide range of tasks,

2) using of specialized semantic indexing systems based on the multi-agent paradigm.

Let us consider each of these methods. Most popular agent platforms are JADE [1], MASDK [2], Zeus [3].

TABLE I. GENERIC AGENT-BASED PLATFORMS

	JADE	**MASDK**	**ZEUS**
Developer community	Telecom Italia Lab	SPIIRAS	BT Laboratories
License	LGPL	LGPL	LGPL
Description	This is the platform for rapid development of multi-agent systems, which implements FIPA standards [4]. JADE provides base classes for creating agents and infrastructure for the operation of multi-agent system.	This is the software environment for multi-agent application development that supports the full life cycle application development of MAS. The agent platform, which is the part of MASDK, works on the principle of P2P.	This is the agent platform designed for rapid development of multi-agent applications. Zeus provides a library of agent components.
Description of the agent behavior	Set in the code of the agent class that inherits from Agent.	Set with language ASML. This language is used for generating applied MAS.	Set in an environment for building agents, from which the agent code is generated.

Each of these agent platforms allows one way or another to describe the behavior of the agent. Depending on the platform we can define almost any behavior of an agent, programming or describing it using specific language. So we can determine the behavior of the agent that implements mechanisms of semantic indexing. The key problem of this approach is the high overhead of run-time. This is due to a complex infrastructure applications received applications. This can be compared with a programming in high level language and Assembler. The actions are the same, but the performance is significantly different. Therefore, such an approach to the problem is not satisfactory.

As noted above, the second approach to the problem of semantic indexing is the use of specialized semantic indexing systems based on the multi-agent paradigm. In this area, it was found only one solution – Magenta Toolkit [5]. This software solution is commercial, so there is no legal possibility to evaluate the effectiveness of work and, especially, to study the mechanisms of their internal functioning. Magenta Toolkit developers have written a number of publications [6, 7], which describe the principles of the system in outline without specifics. This decision is also not satisfactory.

Therefore, the task of the research is development of an open (open source and detailed descriptions of the principles) and an effective method of semantic indexing based on the multi-agents paradigm. In addition, you also need the option to apply this method to all electronic records. So the agent platform must be developed.

III. DOCUMENT ANALYSES STEPS

On Fig. 1 text mining process steps are shown. Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar.

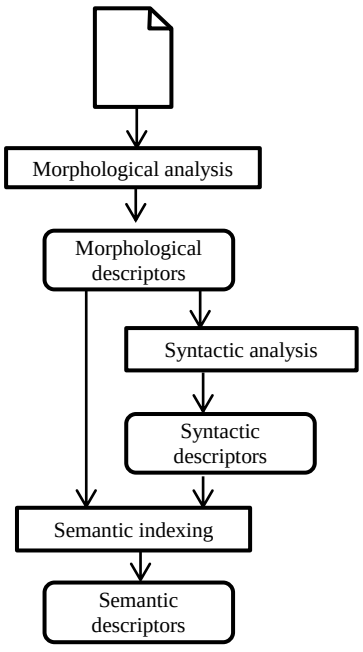


Figure 1. Steps of document analyses

Simplifying the problem we assume that first two steps of text mining process have been made, i.e. a set of syntactic and morphological descriptors for each sentence have been obtained. The result of semantic analysis (indexing) is a semantic descriptor of text that binds the syntactic descriptors of sentences to the elements of the domain ontology which is used for semantic search.

Descriptors (morphological, syntactic, and semantic) are a set of tags which marks words in the sentence. Syntactic and morphological descriptors will be put into relational tables for two reasons. Firstly, syntactic and morphological descriptors will be actively used for semantic indexing. Secondly, we don't want to pile up document by tags. Each word in the text (except for a different kind of stop words) will be assigned a unique identifier. Each identifier corresponds to a separate table row.

Thus, *i*-th row of the table looks like $(id_i, \{a_j\}_i)$, where id_i – the identifier of the word, $\{a_j\}_i$ – set of attributes (tags) that have been assigned to a given word during morphological and syntactic analysis process. In each row of syntactic descriptor table an identifier of applicable syntactic rule is indicated. The syntactic rule is a rule for constructing syntactically correct sentences. The semantic descriptor is represented as set of tags (semantic markup) within the indexed document.

IV. AGENT-BASED SOLUTION

Further let us consider the process of building a semantic index based on multi-agent approach (see Fig. 2).

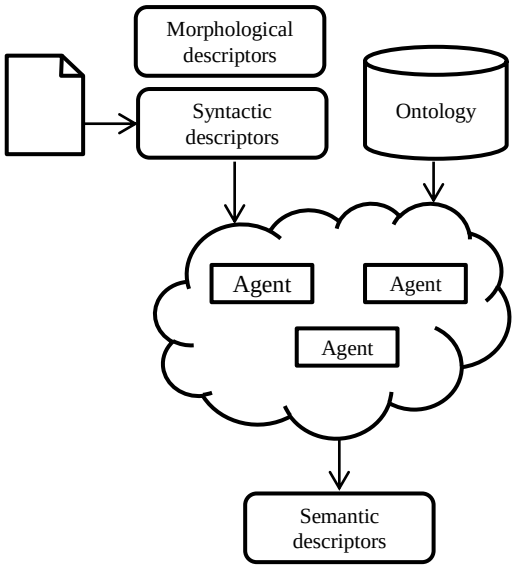


Figure 2. Arhitecture of agent platform

Agents have access to a domain ontology, syntactic, morphological descriptors and electronic documents which will be indexed. Indexing process is produced on the sentences in the text. Sentences are processed sequentially by agents. The agents form a "team" to index the particular sentence. Thus, agents in the system after the start of the indexing are divided into teams.

A. Agent Types

The following types of agents are identified in the system, according to the functional separation:

1) Team Lead First Level Agent - TLFL agent,
2) Team Lead Second Level Agent - TLSL agent,
3) Word Indexer Agent - WI agent,
4) Index Writer Agent - IW agent.

The task of WI agent is accessing to the domain ontology and obtaining the set of possible semantic tags for the indexed word. An input word is passed to the WI agent for indexing with the parameters obtained at the stage of morphological and syntactic analysis. Resulting set of possible semantic tags is passed to the TLSL agent.

TLSL agent binds to syntactic and morphological descriptors of the sentence and distributes words to all available WI agents. TLSL agent finishes its work on the sentence when the consistent semantic descriptor is formed and written to the document. TLSL agent plans actions for the WI agents and also participates in the auction for the resolution of contradictions. After building a consistent semantic descriptor TLSL agent transmits the generated semantic descriptor of the sentence to IW agent who writes semantic tags to the document.

TLFL agent binds to syntactic, morphological descriptors of the document and distributes descriptors of the sentences to all available TLSL agents. TLFL agent monitors the work of TLSL agents. If the work on the sentence is completed TLSL agent gives TLFL agent a new sentence. In addition, TLFL agent conducts an auction among TLSL agents to resolve ambiguity in the descriptors (see details in section «Agent negotiation»).

B. Agent communication

Agents communicate through language FIPA ACL (Agent Communication Language developed by FIPA) [8]. Two types of actions are used. They are inform (inform about anything) and perform (execution of an action).

Inform action type is implemented in the following cases:

1) WI agent informs the TLSL agent of completion of indexing word and give it the set of possible semantic tags; content of the communication is as follows: (id, tags), where the id is the identifier word that came to be indexed, tags are returned set of possible semantic tags;
2) TLSL agent informs the TLFL agent of completion of indexing sentence with a specific identifier; content of this message contains an identifier of indexed sentence.

Perform action type is implemented in the following cases:

1) TLFL agent gives to the TLSL agent a task to index a sentence with a specific descriptor; content will look like this: (id, descriptor), where the id is the identifier of the sentence, descriptor is descriptor of the sentence received as a result of syntactic and semantic analysis;
2) TLSL agent gives a task to the WI agent to index a word with specific id; content will look like this: (id, word, parameters), where id is ID of the word, word is the word for indexing, parameters are parameters obtained at the stage of morphological and syntactic analysis;
3) TLSL agent gives a task to the IW agent to write semantic tag of specific word; content is as follows: (word, tag), where the word is an indexed word, tag is just a semantic tag of indexed word.

C. Planning

The planning is dynamic. TLSL agents themselves form a team of agents from the available WI agents. A count of needed WI agents depends on structure of a sentence. With a lack of WI agents at the time of formation of the team TLSL agent may designate to perform indexing of few words at once to the same WI agent. TLFL agent monitors the performance of work of TLSL agents and if they are released it assigns them new sentences for indexing. Completing of work of the agents (WI and TLSL) monitored not only by sending their corresponding messages of inform type, but also change their states (agent states) in the meaning of "vacant."

D. Agent knowledge bases

WI agents and IW agents are primitive reflex agents working in the mode of stimulus-response. Their main function is a simple, no inference, execution of work. In the knowledge bases of these agents are only procedural steps. Knowledge bases of TLFL and TLSL agents represent productions with embedded procedural actions. In fact, the script actions are necessary for the distribution of work between agents. Accordingly TLSL agent knowledge base contains a script for word distribution among WI agents, and TLFL agent knowledge base includes a script for sentences distribution between agents TLSL.

E. Agent negotiation

TLFL agent conducts an auction among agents TLSL, each of which has a contextual memory (training component). Every TLSL agent using the contextual memory votes for a one option of sematic descriptor of the sentence. Option of semantic descriptor of the sentence with the highest number of votes shall be considered as a true semantic descriptor of the sentence. The set of all consistent semantic descriptors of the sentences form the document semantic descriptor.

V. Conclusion

So, in this paper we have discussed various approaches to solving the problem of document semantic indexing based on multi-agent paradigm. We propose a variant of the solution of that problem and describe it in terms of morphological, syntactic and semantic descriptors of the text. Specialized types of agents are introduced and the general principles of multi-agent system functioning are described.

References

[1] JADE Programmers guide. http://sharon.cselt.it/projects/jade /doc/programmersguide.pdf

[2] Gorodetsky V., Karsan O., Samoilov V., Serebryakov S. "Applied multi-agent systems of group control" / / Artificial intelligence and decision making № 2.2009

[3] ZEUS Techical Manual. www.upv.es/sma/plataformas/zeus/Zeus-TechManual.pdf

[4] The Foundation for Intelligent Physical Agents. http://www.fipa.org

[5] The Magenta Toolkit. http://www.magenta-technology.ru/ru/

[6] Andreev V., Iwkushkin K., Karyagin D., Minakov I., Rzevski G., Skobelev, P., Tomin M.: Development of the Multi-Agent System for Text Understanding. In 3rd International Conference 'Complex Systems: Control and Modelling Problems'. Samara, Russia, September 4-9 2001, 489 – 495.

[7] Minakov I., Rzevski G., Skobelev, P., Kanteev M., Volman S. : Multi-Agent Meta-Search Engine Based on Domain Ontology. http://www.magenta-technology.ru/ru/

[8] FIPA ACL Message Structure Specification. http://www.fipa.org/specs/fipa00061/SC00061G.pdf

One approach to metadata inclusion in electronic documents

Vyacheslav Bessonov
Computer Science Department
Perm State University
Perm, Russia
v.bessonov@hotmail.com

Viacheslav Lanin
Department of Business Information Technologies
National Research University Higher School of Economics
Perm, Russia
lanin@perm.ru

The article describes an approach to the metadata inclusion into Open XML and ODF documents. This metadata allows implement semantic indexing. The described solution is realized as a software library SemanticLib that provides a uniform access to documents in these formats.

Open XML; OpenDocument Format; metadata; RDF

INTRODUCTION

Semantic indexing of electronic documents is intended to include special structure associated with the content of documents in its metadata. Most of the currently used electronic document formats do not permit the inclusion of additional information. Electronic documents open formats Office Open XML and OpenDocument Format become increasingly popular nowadays. By author's opinion these formats are the most promising.

I. OFFICE OPEN XML FORMAT

Office Open XML (OOXML) is a set of open formats based on ZIP and XML technologies intended for representation of electronic documents package of office applications such as spreadsheets, presentations, text documents.

In 2006 the Office Open XML was recognized as the standard ECMA-376 and 2008 as the international standard ISO/IEC 29500:2008.

Since 2007 version of Microsoft Office OOXML is the default format for all applications included in the package of Microsoft Office.

For each document type its own markup language is used:

• WordprocessingML for text documents;

• SpreadsheetML for spreadsheets;

• PresentationML for presentations.

OOXML also includes a set of specialized markup languages that can be used in documents of various types:

• Office Math Markup Language is used to represent mathematical formulas;

• DrawingML is used to represent vector graphics and diagrams.

Office Open XML uses Open Packaging Convention (OPC), created by Microsoft and intended for storing a combination of XML and binary files (eg, BMP, PNG, AVI and etc.) in a single container file.

II. OPENDOCUMENT FORMAT

OpenDocument Format (ODF) is an open document file format intended for storing and exchanging editable office documents such as spreadsheets, text documents and presentations.

ODF standard is created and supported by Committee ODF Technical Committee organization OASIS (Organization for the Advancement of Structured Information Standards). OASIS published ODF 1.0 in May 2005, Commission International Organization for Standardization / International Electrotechnical Commission ratified it in May 2006 as ISO/IEC 26300:2006, so ODF become the first international standard for office documents.

ODF was accepted as the national standard in the Russian Federation, Brazil, Croatia, Italy, Korea, South Africa, Sweden and Venezuela.

III. APPROACH DIFFERENCES

Although both formats are based on open technologies, and are actually ZIP-archives that contain a set of XML-files defining the contents of the documents, they use very different approaches to solve the same problems and have radically different internal representation.

Format ODF reuses existing open XML standards, and introduces new ones only if it is really necessary. For example, ODF uses a subset of Dublin Core to represent document metadata, MathML to present mathematical expressions, SMIL to present multimedia content of the document, XLink to provide hyperlinks, etc. It means primarily it is easy to use this format by people already familiar with the existing methods to process XML.

The Office Open XML Format uses solutions developed by Microsoft to solve these problems, such as, Office Math Markup Language, DrawingML, etc.

IV. Office Open XML and OpenDocument Format APIs

As mentioned above, despite the same set of used technologies – XML and ZIP, Office Open XML Format and the OpenDocument Format have very different internal representation. Besides over the formats are under permanent development, there are currently several revisions of each format with very different possibilities.

For the Office Open XML they are:

• ECMA-376;

• ISO / IEC 29500:2008 Transitional;

• ISO / EC 29500:2008 Strict.

For the OpenDocument Format they are:

• ISO / IEC 26300;

• OASIS ODF 1.1;

• OASIS ODF 1.2.

Existing software solutions designed to work with this formats are quite different. We will consider some of them.

A. Office Open XML APIs

All libraries and other software tools for working with documents in the Office Open XML Formats can be divided into two broad categories. We will reference these technologies next way:

• OPC API – low-level API, allowing working with OPC-structure of OOXML documents, but not providing opportunities to work with markup languages Office Open XML. Examples of those APIs are shown in Table I.

• OOXML API – high-level API, designed to work with specific markup languages (WordprocessingML, SpreadsheetML, PresentationML). Libraries and tools of this category typically are based on OPC API. Examples of OOXML APIs are shown in Table II.

TABLE I. OPC APIs comparison

	ECMA-376	ISO/IEC 29500:2008	ISO/EC 29500:2008 Strict
Packaging API	+		
System.IO.Packaging	+		
OpenXML4j	+		
libOPC		+	

TABLE II. OOXML APIs comparison

	ECMA-376	ISO/IEC 29500:2008	ISO/EC 29500:2008 Strict
Microsoft Office 2007 Automation		+	
Microsoft Office 2010 Automation			
Open XML SDK 2.0	+		
Apache POI		+	

B. ODF APIs

Libraries for operating with electronic documents in the ODF format can be divided into two broad categories too:

• Libraries in the ODF Toolkit. ODF Toolkit Union is the community of open source software developers. Its goal is simplifying document and document content software management.

• Third-party organizations libraries.

TABLE III. ODF APIs comparison

	ISO/IEC 26300	OASIS ODF 1.2
AODL		
odf4j	+	
ODFDOM	+	+
Simple Java for ODF		
lpOD	+	

V. SemanticLib

It is obvious that there should be a universal approach, allowed to work with electronic documents in various formats in a standardized way. SemanticLib was developed to solve this problem.

SemanticLib - is a program complex designed for semantic indexing of electronic documents. SemanticLib main functions are:

• create new and edit existing Office Open XML and OpenDocument Format documents;

• work with the document metadata, linking the metadata with the content of the document;

• providing an interface for SPARQL queries to the metadata document.

Here are the basic components of SemanticLib:

• *SemanticLib DOM* is an abstract model of an electronic document and its metadata, which may be applicable for the description of electronic documents in various formats (Office Open XML, OpenDocument Format).

• *SemanticLib Plugins* are specific SemanticLib DOM implementations using specialized API. For example, the OpenXmlSdkPlugin uses Open XML SDK 2.0, a plugin OdfDomPlugin uses ODFDOM [6].

• *SemanticLib Interpreter* is a module that allows to work with SemanticLib online.

• *SemanticLib Document Browser* is a GUI application that allows you to analyze the structure of electronic documents, view its metadata and run SPARQL queries.

• *SemanticLib Shell Extension* is Microsoft Windows Explorer extension, which adds to its context menu extra

points, allowing to run SemanticLib Document Browser for certain types of documents (.docx, .odt, etc).

VI. SemanticLib DOM

SemanticLib DOM is a set of interfaces and abstract classes that describe the model of an electronic document and its metadata. This model was designed in accordance with the ISO/IEC 29500 standard, which described in [1], [2], and the OASIS ODF 1.2 specification, which described in [3], [4]. Software implementation of this model is based on the implementations used in Open XML SDL 2.0 and ODFDOM libraries.

SemanticLib DOM as well as the Open XML SDK 2.0 and ODFDOM has layered architecture:

• the first layer contains functions for working with document package (e.g., OPC package as described in [2], or ODF package as described in [4]).

• the second layer contains features designed specifically to work with the structure of the document: add/delete paragraphs, change document content, etc.

A. *Document structure*

The document model has a hierarchical structure and schematically depicted in Fig. 1.

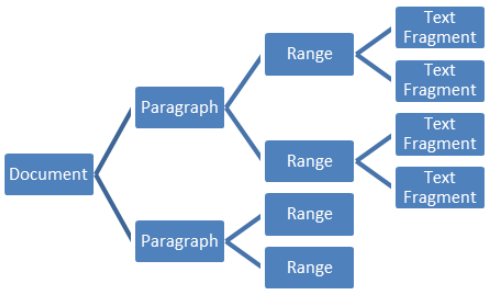


Figure 1. The model of the document used in SemanticLib

Fig. 2 shows a software implementation of DOM SemanticLib.

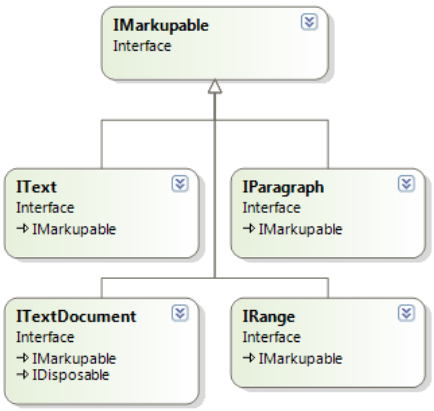


Figure 2. SemanticLib.Core.dll interfaces to work with OOXML and ODF documents

IMarkupable interface contains properties and methods that are used for semantic markup.

ITextDocument interface contains methods and properties for working with text documents, presented in a format like OOXML, and in the format ODF.

IParagraph interface contains properties and methods for working with particular paragraphs of the document.

IRange interface is used for working areas with continuous text contained in paragraphs.

IText interface is designed to work with particular text fragments contained in the text fields. The reason for the separation is the necessary to provide an opportunity for semantic markup of particular words in a text document.

It is worth to note that all mentioned interfaces inherit from interface IMarkupable, so the semantic markup can be used as well as at the level of the document and to its particular elements such as paragraphs, text fields and text fragments.

It was mentioned that a text document and its fragments are containers, i.e. they contain other elements:

• a text document contains a collection of paragraphs;

• each section contains a collection of text fields;

• each text area contains a collection of text fragments.

Fig. 3 shows the hierarchy of abstract classes that represent collections of text documents.

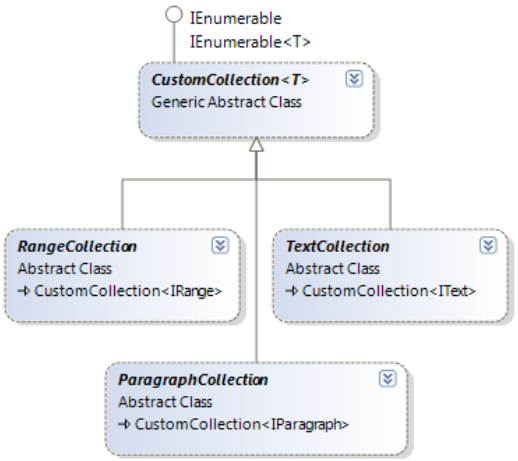


Figure 3. Collections of DOM SemanticLib

CustomCollection is the base class for all collections SemanticLib. It contains the common set of properties and methods, such as, for example, adding a new item in a collection, inserting a new item in a collection, removal element of the collection, etc.

ParagraphCollection represents a collection of paragraphs.

RangeCollection represents a collection of text fields.

TextCollection represents a collection of text fragments.

B. *Metadata model*

Different types of metadata SemanticLib can be divided into two groups:

• non-RDF metadata

• RDF metadata.

1) *Non-RDF metadata*

Metadata model of the group was developed based on analysis:

• core properties, extended properties, custom properties, described in [1], [2];

• predefined non-RDF metadata elements, described in [3].

Table IV shows a list of supported in the current version SemanticLib DOM properties that allow to describe the metadata of the document.

TABLE IV. NON-RDF DOCUMENT METADATA

SemanticLib Property	OOXML Property	ODF Property
Created	created	creation-date
Creator	creator	initial-creator
Description	description	description
Keywords	keywords	keyword
Language	language	language
LastModifiedBy	lastModifiedBy	creator
LastPrinted	lastPrinted	print-date
Modified	modified	date
Revision	revision	editing-cycles
Subject	subject	subject
Title	title	title
Application	Application	meta:generator
Characters	Characters	meta:character-count
CharactersWithSpaces	CharactersWithSpaces	meta:non-whitespace-character-count
Lines	Lines	meta:row-count
Pages	Pages	meta:page-count
Paragraphs	Paragraphs	meta:paragraph-count
Template	Template	meta:template
TotalEditingTime	TotalTime	meta:editing-duration
Words	Words	meta:word-count

2) *RDF metadata*

SemanticLib RDF metadata model is based on the ODF 1.2 metadata model, as described in [3].

Access to all RDF metadata of the document is performed by the manifest metadata, which in turn is also RDF document. At the program level for this interface IMetadataManfiest is used for this purpose. With IMetadataManifest you can access directly to the manifest's RDF graph, to gain access to existing or to add new RDF metadata files.

SemanticLib uses for work with RDF an Open Source Library dotNetRDF. It provides an opportunity to work with RDF graphs in memory, to serialize/deserialize graphs and to run SPARQL queries.

VII. SEMANTICLIB PLUGINS

The SemanticLib core library contains only a description of the document model (DOM). Implementation of the methods for processing documents of any format is contained in the plug-ins. Typically each plug-in is an implementation of SemanticLib DOM with some libraries described in paragraph V. For example, a plug-in SemanticLib.OpenXmlSdkPlugin.dll uses API Open XML SDK 2.0, a plug-in SemanticLib.LibOpcPlugin.dll contains API libOPC.

Using plug-ins using makes possible a high degree of flexibility and extensibility. If a library expire or a new one appears, developer can just replace or add a plug-in without changing the basic functions of libraries and existing code.

However, plug-in development becomes significant difficult because of the existing the variety and diversity libraries. For example, the library Office Open XML SDK 2.0 is created on the platform .NET, while the library ODFDOM is created in Java, which means a significant difficulty trying to promote interoperability between these libraries. It is also difficult to ensure interoperability between C/C++ and .NET libraries. Let's consider how these issues are resolved in SemanticLib.

A. *C/C++ plug-ins*

Let's see the interoperability between C/C++ and .NET code by the example LibOpcPlugin, which is the implementation of DOM SemanticLib with libraries libOPC, written in ANSI C.

It was decided to use C++/CLI to enable interoperability between managed and unmanaged code. The main advantage of this solution is the ability to use object-oriented programming style even interacting with procedural code of libOPC. In this case plug-in consists of a set of classes that implement the interfaces of DOM SemanticLib.

B. *Java plug-ins*

Interoperability between Java and .NET code will be considered on the example plug-SemanticLib.OdfDomPlugin.

There are some solutions to ensure interaction between Java and .NET applications. For example, there are products of JNBridge company, which provide both in-process and inter-process (network cloud) communication.

However, in SemanticLib Open Source project jni4net was selected. Its aim is providing an in-process communication.

Deal with jni4net has several stages:

• Creating a proxy for a Java library with a special utility proxygen, which is part of jni4net.

• Creating a .NET stub, which provides the work with Java-proxy. This step is also performed using proxygen.

• Implementation of a plugin functional using the resulting stub.

This process is quite complex and requires specific skills, so it is necessary to create automation tools in future versions of SemanticLib.

C. *Working with plug-ins*

One of the significant advantages offered by SemanticLib, is the ability to work with a dynamic plug-ins. This feature is important if you work with a large number of different plug-ins. Plug-in manager, which is part of SemanticLib, helps user to manage the plug-ins loading process. Plugin Manager provides the following features:

• Find the required plug-ins in accordance with certain criteria, such as the name of the plug-in or the format of the document.

• Loading and unloading plug-ins.

• Viewing the meta-information about the loaded plug-ins (name, manufacturer, document format, etc.).

VIII. SEMANTIC LIB INTERPRETER

The main usage scenario SemanticLib involves writing code in one of the support. NET languages and the subsequent compilation of the code. However, this approach is not always convenient. Especially it is not convenient for users who are not programmers and have no special skills to work with IDE, compilers, etc.

SemanticLib Interpreter has been developed to solve these problems. He adds two more SemanticLib use cases:

• interactive work in interpreter mode;

• writing scripts and their subsequent dynamic compilation without the need for third-party tools (IDE, compiler, etc.).

However, it should bear in mind that SemanticLib Interpreter is a DLL and it need a host CUI or GUI application (for example SemanticLib Document Browser).

A. *Interpreter mode*

In this mode user interacts with SemanticLib in progressive interpretation kind. In this case the interpreter correctly handles all the variables, i.e. following line:

```
“var plugin = PluginManager.FindPlugin("SemanticLib.OpenXmlSdkPlugin.dll")"
```

is absolutely correct and the variable “plugin” can be used in subsequent commands.

B. *Scenario mode*

In this mode, the user instead of the progressive interpretation creates a so-called "scenario" (set of SemanticLib commands) and then compiles them into a CUI application.

IX. SEMANTICLIB DOCUMENT BROWSER

This application is still under development. It should be a GUI application written using Microsoft Windows Presentation Foundation framework. The main functions of this applications are:

• view the document structure and it’s metadata;

• SPARQL queries execution and presentation.

X. RESULTS

• SemanticLib DOM library that allows you to perform basic operations with the structure of Office Open XML and OpenDocument Format documents.

• Metadata model and that allows to manipulate document-level metadata.

• SemanticLib.OpenXmlSdkPlugin and SemanticLib.OdfDomPlugin plugins.

• SemanticLib Interpretet that allows you to use SemanticLib functions in interactive mode.

XI. GOALS

• To refine SemanticLib DOM, so that it covered more fully the ISO/IEC 29500 standard and OASIS ODF 1.2 specification.

• To refine the metadata model of the document, in particular the binding metadata to the document content.

• To develop a mechanism for context queries, i.e. queries that would take into account the document contents.

• To develop SemanticLib Document Browser and SemanticLib Shell Extension.

CONCLUSION

Semantic indexing of documents in Open XML Formats and Open Document Format can be implemented on the basis of the described solutions. The developed library is a part of the intelligent document processing project, but also can be used to solve other problems that require metadata inclusion.

REFERENCES

[1] ISO/IEC 29500-1 Second edition, 2011-08-15. Information technology – Document description and processing languages – Office Open XML File Formats. Part 1: Fundamentals and Markup Language Reference.

[2] ISO/IEC 29500-2 Second edition, 2011-08-15. Information technology – Document description and processing languages – Office Open XML File Formats. Part 2: Open Packaging Conventions. 138 c.

[3] Open Document Format for Office Applications (OpenDocument) Version 1.2 Part 1: OpenDocument Schema 29 September 2011.

[4] Open Document Format for Office Applications (OpenDocument) Version 1.2 Part 3: Packages 29 September 2011.

[5] OASIS OpenDocument 1.2 Metadata Examples, Oct 2,2009.

[6] http://incubator.apache.org/odftoolkit/odfdom/index.html

[7] http://dotnetrdf.org/

Data mining techniques in Real-time Marketing[1]

Vladimir Gromov
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
v.gromov@hotmail.com

Scientific Advisor: Prof. Sergey Avdoshin
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
savdoshin@hse.ru

***Abstract* – This paper gives an overview of the concept of a new system to support CRM in real-time using data-mining techniques. To ensure that in the modern world of dynamic companies remain in the leaders of their industry they need to continually monitor activity of their customers. Such activities are performed by analysts. Nevertheless, people are unable to handle huge amounts of data, which are encountered by such organizations as banks or mobile operators daily. In this situation information systems come to help. Software Engineering Department Higher School of Economics, in collaboration with IBM Company is conducting research in this area; the interim results have been examined in this work.**

I. Introduction

Different people use different products and services for different reasons. However, they are all part of the company's customer base and demand a handling that best suits their characteristics. Nowadays companies tend to build long-term relationship with the customers. In order to maintain such relationship companies need to discover the customer's needs. However today companies often have huge customer base, so it is unprofitable to handle each customer apart from others. Another problem is that if company wants to promote some specific product among its customers, including all customers into the target group would be either expensive for a company, or irritating for them, so they may turn to competitors.

In such situation data mining comes to help. With aid of historical data company can identify features of customers that lead to acceptance of some product or offer so that it would be possible to narrow the targeted audience. It could be also useful identify group of customers that require specific handling in order to ensure their loyalty.

II. Current situation in CRM

Nowadays marketing is automated mainly with use of CRM systems. There are several types of CRM systems, but we are interested in the analytical systems.

They provide the following capabilities:

- Classification of customers by some basis
- Analysis of market situation and competitors
- Analysis of choice and price of goods
- Analysis of conducted sales
- Analysis of purchases and supplies
- Accounting and evaluation of marketing campaigns

However, traditional systems are unable to apply data mining techniques in order to make predictions about customer's behavior and aid company in making marketing decisions. Moreover they are unable to handle huge amounts of data in real-time. The proposed solution is aimed at overcoming these issues.

III. Proposed methods

The proposed method implies usage of several types of models [2]:

- Response Modeler — Increases ROI on acquisition campaigns by identifying and allowing to target those prospects who are most likely to respond to direct mail campaign.

[1]This work is being performed within the scope of the research on the topic "Research and development of innovative unifying models of intelligent systems for the situational response and safety control on the Russian railways", state contract 07.514.11.4039 on September 26, 2011 at lot № 2011-1.4-514-045 "Development of algorithms and software systems for solving problems of exceedingly large scientific data sets storage and processing and data streams collection in real-time" as part of the federal target program activity 1.4 " Research and development in Russian scientific-technological system 2007-2013 evolution priority directions".

- Cross Seller — helps maximize sales of products and services to existing customer base by identifying those customers that are most likely buy other products and services (e.g., additional services or products based on related product and service purchases) as well as the specific products/services a particular customer would be most interested in.
- Segmenter and Profiler — helps analyze and better understand customers for more one-on-one marketing by segmenting them into homogeneous groups (using natural clustering methods, segmentations generated from response, cross-sell or customer valuation models, or using manually defined segmentations) and profiling them. The results of these groups can be used to enhance customer acquisition programs (by finding prospects that are similar to existing customers) as well as retention programs (by making a focused offer that will appeal to a specific group).
- Customer Valuator — predicts the spending level or profitability of your customers over a specific time period to help forecast demand, strategize acquisition campaigns, and identify most valuable customers. These models can be used in many ways to optimize marketing efforts.

Provided the data mining models are properly built, they can uncover groups with distinct profiles and characteristics and lead to rich segmentation schemes with business meaning and value [1].

IV. A Customer Scenario for the Proposed System

Let us consider a customer that wants to offer advertisements to consumer who visits CNN.com.

We have several advertisements at the disposal for offer. So, our extreme opportunities are:

- Bid low on everything
- Bid high on only those you are most confident of a hit.

The second option is available if we could build up an "anonimized profile" of a consumer based on:

- Transactions from this customer
  - Cardholder since YYYYMM
  - Average transaction value
  - Monthly transaction value
  - Categories purchased
  - Brands purchased
- Descriptive
  - Age
  - Gender
  - Family situation
  - Zip code
- Interactions
  - Web registration
  - Web visits
  - Customer service contacts
  - Channel preference
- Attitudes
  - Satisfaction scores
  - Shopper type
  - Eco score

All these data allow us identify consumer as well as predict his response on one or another advertisement. The data will be processed in real-time and decision will be made once consumer enters the site. All decisions are considered and used for further scoring.

So now let us assess the how such system can be implemented using the sample data.

V. System Structure

The sample data represents statistics of bank loans decisions. There is no information about input data, as it is covered and changed to meaningless symbols and number. The only field we have information about is final decision: positive or negative.

First of all, a model is built in Modeller that encapsulates the above mentioned techniques (fig. 1).

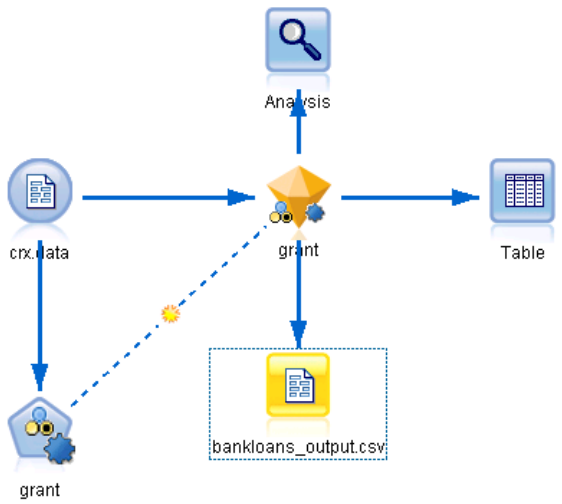


Figure 1. SPSS Modeller model

On the figure 1 there is a model that consists of data source node that imports data into model.

After that auto classifier comes to work. It uses several data mining techniques and selects 3 decision trees that provide highest confidence. In future scoring will be performed and prediction will be maid according to the prediction with the highest confidence.

After that the model is trained on some historical data. Another portion of historical data is used for evaluation of obtained model.

Next step is exporting of model and integrating it into IBM InfoSphere Streams operator (fig. 2). IBM InfoSphere Streams is a runtime environment that allows easy distribution of load between nearly unlimited number of computational nodes.

Figure 2. IBM InfoSphere Streams application graph

On figure 2 there is an application graph that denotes structure of InfoSphere Streams program. In this application data is read from an input file, than it is passed to an operator that applies SPSS Modeller model to the data and finally scoring results are put to an output file. The model is executed in the Streams program by special operator that calls the SPSS Modeller Solution Publisher, with the help of special API.

The training never stops. Every time new data arrives, model is trained once again. So, each time a transaction is made by customer, the system estimates whether he is likely or not to accept some marketing proposal and depending on estimation makes offer.

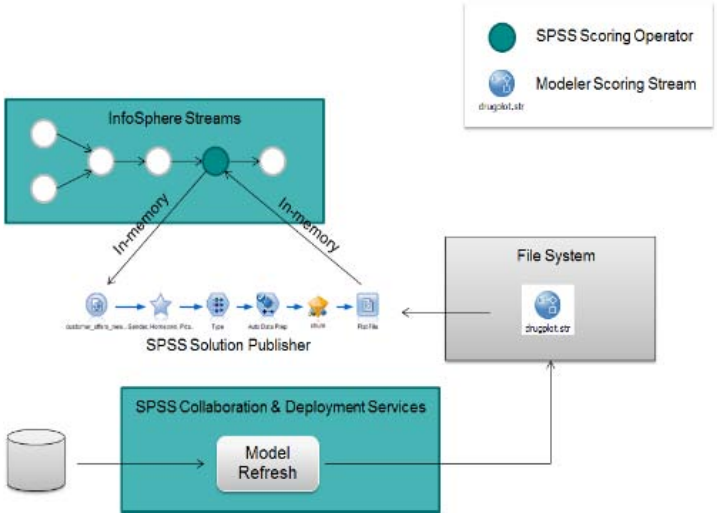


Figure 3. Operation of model.

As it is seen from the figure 3, the model is refreshed in real time by SPSS Colloboration and Deployment Services.

VI. CONCLUSIONS

In this work data-mining techniques and information system implementing these techniques are presented. There are some tasks that are already implemented, like integration of Infosphere Streams and Modeller. The next step would be identification of set of models that would be used to analyze input data and transform it to output. The architecture of Infosphere streams allows to split the system among several computational nodes and thus it can be easily scaled in order to meet business needs.

REFERENCES

[1] K. Tsiptsis and A. Chorianopoulos, *Data Mining Techniques in CRM* 1st ed., John Wiley & Sons, Ltd. 2009

[2] Ing. Vladek Šlezingr. Campaign Management system Technical Proposal For: Home Credit International a.s. 2011

Multistroke Mouse Gestures Recognition in QReal metaCASE Technology

Maria Osechkina
Mathematics and Mechanics Faculty, SPbSU
Saint-Petersburg, Russia
osechkina.masha@gmail.com

Yuri Litvinov
Mathematics and Mechanics Faculty, SPbSU
Saint-Petersburg, Russia
yurii.litvinov@gmail.com

Timofey Bryksin
Mathematics and Mechanics Faculty, SPbSU
Saint-Petersburg, Russia
timofey.bryksin@gmail.com

Abstract - **Approaches for multistroke mouse gestures support are considered in this paper. Presented way of gestures implementation is oriented on creation of new elements on a scene or diagram in meta-CASE system. We propose to generate examples of mouse gestures for elements and to allow user create elements by fast mouse move. We present numerical comparison of gestures recognition algorithms before and after training by k-means algorithm. The proposed approach is implemented in QReal meta-CASE-system.**

Keywords – recognition, multistroke gestures, meta-CASE-system, k-means algorithm

I. INTRODUCTION

World practice has gained considerable positive experience with domain-specific visual techniques for solving problems of industrial software development. In comparison to the traditional approach of "manual" coding we observe 3-10 times increase in productivity on average [4].

There is no significant gain in programmers productivity when using general purpose visual languages such as UML, and therefore domain-specific visual languages become more and more popular. Meta-CASE-systems are designed for quick creation of languages for particular subject areas and generation of tool support for these languages. One example of meta-CASE-systems is QReal [5, 6], which is developed at Software Engineering chair of St. Petersburg State University. To make rapid development of new visual editors in QReal possible a meta-editor was developed. It allows to create metamodels of new visual languages by describing objects on a diagram and associations between them, and defining visual representation of these elements. Then the created metamodel is compiled into a dynamic library and is plugged in QReal in run-time.

The effectiveness of each tool is determined by how easy and fast it performs operations which this tool is intended for. In modeling some of the most frequent operations with objects and relationships in diagrams are their creation and deletion. In many of existing CASE-tools to create a desired object on a diagram one must first either find it on a toolbar or select it from a menu, and then specify position on a diagram where this element sould be placed. Many toolkits provide ability to create an element drag-and-dropping it from a palette. The problem is that the number of diagram types and objects in the palette of each diagram can be quite large (for example, 13 types of diagrams in UML 2.4). It is not always possible to leave on a palette only items that are specific to current diagram, because sometimes there is a need to use elements from different diagrams, for example, for rapid prototyping. Even in case of such basic operations like creation of new element, developer has to make a set of purely mechanical actions and also remember in which tab of a palette or which menu holds desired item. He or she has to constantly switch from thinking about the hierarchy of created models to particular questions of usage of selected tool. We believe that the process of adding items to diagrams could be faster and easier if it involved some alternative ways of man-machine interaction.

As an example of such approach of user interface optimization we consider mouse gestures recognition. The main idea is to associate some actions with specific mouse movements performed with some modifier (for example, with mouse button pressed). These actions are executed immediately after the gesture is recognised. It is desirable that there will be no restrictions on the direction of mouse movements, number of clicks, and the sequence of obtained strokes. In case of CASE-system the action associated with gesture could be creation of element on a diagram at a position where the gesture has been performed. For more similarity with natural process of drawing we want to use recognition of multistroke gestures. This approach does not limit the way a gesture is being drawn.

User interfaces based on mouse gestures recognition are widely used not only in CASE-tools, but also in other areas, such as online handwriting recognition in a variety of text editors (for example, for texting in smartphones), creation of objects on forms, diagrams and scenes in different applications, navigation in some web browsers, controlling the character in some computer games. Some utilities (such as StrokeIt, gMote, etc.) even allow you to add support of mouse gestures recognition into an arbitrary application. Moreover, there are CASE-systems that use similar approach, for example, Visual Paradigm. However, support of mouse gestures recognition in such programs is limited a number of fixed gestures. That is unacceptable for scalable systems like meta-CASE-systems. The paper

discribes an experience of implementing mouse gestures recognition mechanism in an application where the complete set of gestures is not known a priori and can be expanded any time. In addition, for most of known algorithms a long process of training is required and a training set is comparable in size with a test set. Because of a requirement of gestures set extensibility we need to reduce time of creation of training sets to a minimum.

II. DEFINITIONS AND ALGORITHM DESCRIPTION

In general, gesture recognition algorithm works as follows: there is a list of ideal gestures, each of which is associated with an action. Ideal gesture is a gesture pattern that is matched to a user action that should be executed after performing this gesture. User performs a gesture holding a mouse button pressed. After a specific signal (e.g. a timeout, a keypress or release of mouse button in case of single mouse stroke gestures), the gesture should be recognized. For that a list of ideal gestures is searched for a gesture similar to the performed one. If such gesture is found, the action that is associated with this gesture should be executed.

Analysing the ideas behind pattern recognition algorithms it becomes clear that before starting recognition we have to build ideal gestures for each action. In QReal this action is creation of a new element on a diagram. It would be convenient if user gesture would correspond to graphical representation of created object. We decided to generate list of ideal gestures based on this consideration. Graphical representation of each object in QReal is stored in XML files in SVG-like graphical format. This format consists of basic elements like line segments, circles, arcs etc. For line segments are defined by coordinates of their ends; circles — by the top-left and bottom-right corners of a circumscribed square with sides parallel to coordinate axes; arcs are defined by top-left and bottom-right corners of a square circumscribed around the circle, a part of which this arc is, and two angles identifting the start of this arc and rotation to the end of this arc. For constructing ideal gesture it is enough to present described simple figures as gesture strokes. Stroke is performed by mouse movement between pressing and releasing of mouse button. Since number of strokes, order of stroke drawing and mouse movement direction has not to be taken into account in the context of recognition problem, ideal gesture can be represented as a list of lines that correspond to segments, circles, and arcs that form object's graphical representation, without specifying beginnings and ends of these lines and the order of their drawing. When user moves mouse, application receives a signal that the mouse is moving with information about its current position. Since these signals are discrete, line drawn by user, as it is received by application, is a polyline, not a continuous smooth curve, as it may seem to the user. This polyline consists of short segments, which ends' coordinates were obtained receiving signals from the hardware. For constructing ideal gesture strokes we have to represent primitive shapes consisting of short line segments similar to polylines received from the hardware. We have to calculate average speed of mouse movement empirically — average distance between two points, coordinates of which were obtained by successive signals from the hardware. Note that this value can be considered as signals rate on the assumption that signals are sent at regular intervals. Segment is presented as stroke by dividing it into equal segments of appropriate length. A circle is represented by an inscribed polygon with a sufficiently large number of sides. Arc is represented by a polygin inscribed in an arc with equal segments.

III. FEATURE SET

Comparison of lists of points is difficult, since actual list of points received from mouse depends on the hardware, speed of the mouse, stretch of a figure and other factors that are not necessary and sometimes excessive to consider for successful recognition. It is convenient to calculate some features, e.g. numerical measures that correspond to an object and represent its essential properties. We introduce a feature set for which comparison is made. In a space formed by feature sets we introduce a function F with two arguments, where arguments are the feature sets. Let M be the space of feature sets.

$$F : M \times M \to \mathbb{R}$$

This function determines the similarity between two sets of attributes. For example, it may be a probability of the user drawing a particular gesture, or the distance between feature sets. For every ideal gesture a feature set is calculated for later comparison.

IV. MOUSE GESTURES RECOGNITION ALGORITHM

In general, algorithm of mouse gestures recognition can be divided into several stages.

1) Identification of strokes in mouse path. At this step a list of stroke point coordinates is being built. These points are obtained from hardware signals like mouse position when the button was pressed, coordinates of mouse movement with button pressed and position of mouse button release.

2) Path filtering. As a result of user's hand shakes a gesture can become distorted. It is desirable to eliminate this distortion before proceeding to gestures comparison. We have to smooth those parts of the stroke that the user meant to draw as straight lines, and don't smooth those, there the user meant to change movement direction. As a general rule, when a user moves the mouse slowly, he typically wants the movement of the mouse pointer on the display to follow the exact path of the mouse, whereas when he or she

moves the mouse more quickly, the main concern is typically where the pointer ends up, not the exact path to get there. Guided by this consideration, smoothness of the mouse path can be based on mouse velocity.

3) Building of feature vector for user's gesture.

4) Selection of an object. We compute the similarity function for feature set related to user gesture, and each of the feature sets that are related to ideal gestures. In our algorithm this function will serve as the similarity distance. We will choose the ideal gesture according to the results, in our case this is ideal gesture, which is associated with the set of attributes to which the distance is at minimum. If there are several ideal gestures which are close to minimum distance, we shall provide user the ability to choose desired object from them himself. If this distance is less than some threshold, the gesture is recognized and we generate corresponding object.

5) Executing of an action. In case of QReal, when an action is executed, new object have to be created on a diagram. The object is generated in such a way that its center coincides with the center of a rectangle circumscribed around user gesture, with sides parallel to the coordinate axes.

There are training algorithms that allow to adjust recognition algorithm's parameters, including a feature set (or vector) that corresponds to the ideal gestures, in order to improve recognition performance. But for such algorithms we need training sets – a database of user gestures where for each gesturewe know a class to which this gesture relates. Creation of such set is mainly manual process that takes time. In QReal editors for new visual languages can be created very fast, and we do not want to slow down the process of language creation by a need of creation of gesture training database manually. As a general rule, in known solutions the number of items in training set is comparable to the number of items in test set. After some experimenting we decided that training is a necessary step, so we would like to reduce training sets size and time spent on training to minimum. To accomplish this it is desirable that percentage of recognized gestures is high even without training.

V. RASTERIZATION

It is obvious that gesture position should not affect recognition. In addition, in QReal you can stretch elements on the diagram along the coordinate axes, so stretch of the gesture should not affect recognition either. For example, the ellipse and the circle should be recognized as the same object. We introduce the feature set, which is invariant to translation and scaling of a gesture: at first, rectangle with sides parallel to coordinate axes is circumscribed around gesture, then the rectangle is divided into equal cells by lines parallel to the coordinate axes. The number of cells is fixed in advance. Then we create a list of cells, which are intersected by the gesture. Coordinates of cells that contain points of the gesture are considered as a feature. A final list of coordinates is considered as a feature set. We call it basic features set. The introduced feature set is invariant to scale and translation: if we stretch or move a gesture, the corresponding feature set will remain unchanged. Having such a set constructed we got rid of unnecessary information about the number of strokes, the sequence of their drawing and direction of mouse movement. Vast majority of user gesture properties that are independent of the above factors, is also persisted for basic feature set, since a basic feature set is essentially a rasterized gesture with low resolution.

VI. FEATURE SETS AND DISTANCE

We would like to introduce the distance between the basic feature sets, which has some geometrical meaning. Let us analyze how we can interpret the similarity of gestures. Note that two gestures can be called similar if for each object (line segment, circle, cell) from the first gesture we can pick similar object from the second gesture so that they are close enough. I.e. for each cell from the first basic feature set we can indicate close cell from the second basic feature set and vice versa. Based on this consideration Hausdorff distance d_H was chosen as distance between the basic feature sets. Let X and Y are basic feature sets, r is distance in the cell space.

$$d_H(A, B) = \max\{X, Y\}$$
$$X = \max_{a \in A} \min_{b \in B} r(a, b)$$
$$Y = \max_{b \in B} \min_{a \in A} r(a, b)$$

I.e. distance r is introduced in the space of cells. Distance from each cell of the first basic feature set to corresponding nearest cell of the second feature set is calculated and vice versa — for each cell of the second gesture distance to the nearest cells of the first gesture is calculated. Hausdorff distance between the basic sets is maximum of calculated distances between the nearest cells.

*A*s distance between cells we considered the Euclidean distance ($l2$), maximum of absolute differences between the coordinates ($l\infty$), sum of absolute differences of the coordinates ($l1$). The best recognition result is achieved by maximum of absolute differences between the coordinates.

Note that basic feature set cardinality is not constant, so for different user gestures feature sets may have different numbers of elements.

There are training algorithms that involve correction of feature vectors corresponding to the ideal gestures. k-means is one of such algorithms. This algorithm finds center of mass for the elements of space, in which the training is executed. Center of mass for a set of vectors is a vector which coordinates are the arithmetic

mean of the corresponding coordinates of vectors from the training set. It is difficult to imagine the addition of basic feature sets and the division by a scalar keeping the geometric meaning of a basic feature set, since the cardinality of the basic feature set is not fixed.

To simplify the training it is desirable to deal with feature vectors, that are feature sets of fixed cardinality, for which the order of the elements is essential. We propose several ways of constructing a feature vector for the basic feature set. In all of the proposed ways we have to construct a matrix, which is interpreted as feature vector. In all cases, the size of the matrix corresponds to the number of cells across the width and height of the rectangle circumscribed around the gesture, for which this basic feature set is built.

The matrix of distance to the basic feature set

M is a matrix constructed by basic feature set and has dimensions $w \times h$, where the dimensions of the matrix are described above. Element of the matrix *M[i, j]* is equal to the distance *r* from cell with coordinates (*i, j*) to the nearest cell of feature set. Thus we have matrix of distances to basic feature set. Note that we can unambiguously reconstruct the basic set of features by constructed matrix. Basic feature set consists of those and only those cells, coordinates of which are equal to coordinates of matrix elements that are equal to 0. The matrix *M* can be represented as feature vector from space $R^{w \cdot h}$ by writing down all of its lines into one. The distance between the feature vectors is equal to norm of the vectors difference. Introduced function (let us call it *F*) is actually the distance between the feature vectors and between basic data sets.

We prove that for the introduced function the properties of distance in space of basic feature sets are true.

A, B – feature sets,

M(A) – matrix corresponding to set *A*

M(B) – matrix corresponding to set *B*

1) $F(A, B) \geq 0$ - is obvious from the properties of norm.

2) $F(A, B) = 0 \Leftrightarrow A = B$

Thus vector norm is equal to 0 if and only if the vector is *(0..0)*, that is $F(A, B) = 0 \Leftrightarrow M(A) = M(B)$

Since the matrix can be uniquely reconstructed by the basic feature set, the basic feature sets are also equal.

$\Leftarrow$ Clear

3) $F(A, B) = F(B, A)$ This equality follows from the properties of the norm.

$$F(B, A) = \|M(B) - M(A)\| =$$
$$= |-1| \cdot \|M(A) - M(B)\| = F(A, B)$$

4) Correctness of the triangle inequality property follows from the norm definition.

Proposition

r is the distance between cells, that is used for calculation of Hausdorff distance between the basic feature sets. Let the same distance *r* be used as the distance between cells for constructing distances to the gesture. The norm in space of feature vectors is the maximum of absolute elements values ($l\infty$), the sum of absolute elements' values (*l1*) or the square root of the sum of the elements' squares (*l2*). Under these condition Hausdorff distance between the basic feature sets is equivalent to the introduced distance between the basic sets.

Proof

Equivalence is transitive. The $l\infty$ (maximum of absolute elements values) is equivalent to *l1* (the sum of absolute elements' values) and to *l2* (the square root of the sum of the elements' squares). I.e. for proof of proposition it is suffice to prove the equivalence of the Hausdorff distance and $l\infty$.

Let *M1* be the matrix that correspond to the first user gesture, *M2* be the matrix that correspond to the second gesture, *FS1* be the basic feature set for first gesture and *FS2* be the basic feature set for second gesture, *dist(FS1, FS2)* be the introduced distance.

$$dist(FS1, FS2) = \max(|M1_{i,j} - M2_{i,j}|) \geq$$
$$\geq \max_{M1_{i,j}=0 \vee M2_{i,j}=0} |M1_{i,j} - M2_{i,j}| =$$
$$= D_H(FS1, FS2)$$

D_H is Hausdorff distance for basic feature sets.

We got that

$$dist(FS1, FS2) \geq D_H(FS1, FS2)$$

We shall prove that

$$dist(FS1, FS2) \leq D_H(FS1, FS2)$$

We consider elements of the first and the second matrix with coordinates *(i, j)*. By construction of matrix next identity is true:

$$M1_{i,j} = min\{r(O, a) \mid a \in FS1\} ,$$

where *O* is the cell with coordinates *(i, j)*. Let minimum is achieved at point *A1*.

Similarly, $M2_{i,j} = min\{r(O, a) \mid a \in FS2\}$

Let minimum be achieved at point *A2*.

Let $r(O, A2) = b \leq r(O, A1) = c$. The case if $r(O, A2) \geq r(O, A1)$ is analogue.

Suppose that there exists a cell *K1* from the first basic feature set, such that $r(K1, A2) < c - b$.

Then by the triangle inequality

$$r(K1, O) \le r(O, A2) + r(A2, K1) < \\ < b + (c - b) = c$$

It is a point *K1* of the first basic feature set, the distance from which to *O* is less than the distance from *O* to *A1*. It contradicts the selection of *A1* as the cell on which the minimum is achieved. It turns out that for each point *A* of the first basic feature set $r(A2, A) \ge c - b$.

It follows from the definition of Hausdorff distance that $c - b \le D_H(FS1, FS2)$. Due to the fact that the cell *O* was selected randomly, $D_H \ge |M1_{i,j} - M2_{i,j}|$ for all *i,j*. Then $D_H \ge \max |M1_{i,j} - M2_{i,j}|$

We got that

$$D_H(FS1, FS2) \ge dist(FS1, FS2) \ge \\ \ge D_H(FS1, FS2)$$

The distances are equivalent, as required.

Note

When vector norm is considered as the maximum of element absolute values, the Hausdorff distance is equal to the introduced distance.

The number of cells in the rectangle

The idea of following algorithm is calculation of the number of cells in a rectangle of $m \times n$, where *m* varies from 1 to the rectangle height, *n* varies from 1 to the width of the rectangle. Element with coordinates (*i, j)* is equal to the number of gesture cells, that are contained in the rectangle $[0, i] \times [0, j]$. If the gesture intersects the same cell several times we count only one intersection. As in the previous case, the matrix is considered as a feature vector from the space $R^{w \cdot h}$, distance between vectors is defined as norm of the vector difference. Note that by construction of the matrix we can unambiquously reconstruct the basic feature set.

Let *e[i, j]* be equal to 1 if there is a cell (*i, j)* belonging to the basic feature set, otherwise *e[i, j]* is equal to 0. Let *J* be constructed matrix, then $e[i, j] = J_{i,j} + J_{i-1,j-1} - J_{i,j-1} - J_{i-1,j}$. We assume that elements of *J* are equal to 0 outside of gesture rectangle. Note that introduced function is indeed the distance between the feature vectors and between basic feature sets, as there is bijection between the space of constructed feature vectors and the space of basic feature sets. Proof is analogue to case of matrix of distance to basic set.

The number of cells in zone

Feature vector used in this algorithm is similar to previous, but we count cells in the zone, not in the rectangle. *J* is matrix $w \times h$. We define element *[i, j]* of matrix as follow: let *d* be a number of cells in the rectangle $(i-1) \times w$, i.e. in the rectangle, which lies entirely under the cell (*i, j)*, *k* is the number of cells in the zone from a number *(i,j)* from *[i, 0]* to *[i, j]*, $J_{i,j} = k + d$. Note that the basic feature set can be unambiguously recovered from the resulting matrix. Then if we introduce a function of distance between the matrices similar to two previous cases, this function will be the distance between the basic feature sets.

Note that all these feature vectors and distance can be combined by considering a new feature set. The distance between combined feature sets is defined as a linear combination of distances with the non-negative coefficients, at least one of which is strictly positive. This linear combination can be considered as distance between combined feature sets.

VII. TRAINING

We can improve gestures recognition with training: if we have user gesture base, and we know which object should be generated for each gesture, we can correct the parameters of recognition algorithm so that other gestures performed by user will be recognized with greater accuracy. There are a lot of algorithms that allow to improve recognition through training set, for example, Neural networks, the method of *k* nearest neighbors, Bayesian method or k-means algorithm.

The neural network is a system of interconnected processes, that are neurons. Training of neural network involves calculating of coefficients for connections between neurons. But a large training set is necessary for successful recognition via neural network. For example, sometimes number of items in a training set is 1.5 times greater than number of items at a test set [2]. As already mentioned, we would like to reduce the training set's size as much as possible, otherwise the generative approach to gestures creation will be meaningless.

We can use the k nearest neighbors method for training [1]. For a gesture feature set we need to find k nearest neighbors among the feature sets that correspond to user gestures in the training set. The gesture is referred to the class to which the majority of the k neighbors is referred. k nearest neighbors is a time-consuming problem in case of multi-dimensional spaces. In addition, in case of a neural network it is sufficient to store only weights of connections after training, and it is unnecessary to store whole training set unless the need of retraining. But this method does not allow to save memory or to compress training set in any way.

Bayesian method [1] is designed for a variety of symptoms, among which there are no statistical dependencies. For each gesture's feature the probability of this gesture belonging to the given class is calculated basing on the training set. Then the total probability of the gesture belonging to the class is calculated. The most probable class is selected as a result of recognition.

Weakness of this algorithm is limitation to the feature set (lack of statistical dependencies).

Idea of k-means algorithm [3] is to correct feature set corresponding to ideal gestures and the maximum threshold distance between feature set corresponding to user gesture and corresponding to the ideal gesture. Feature vector corresponding to ideal gesture is replaced by a center of mass of feature vectors that are corresponded to the gestures of the training set for given object. Threshold distance is recalculated so that distance from the center of mass to each feature vector of training set is less than threshold distance. In this case, after training we can store only corrected feature set that is corresponding to the ideal gesture, and corrected threshold distance.

In case of a neural network, the percentage of recognized gestures would be high after training with a small training set, if only we guessed optimal values of coefficients of connections between neurons good enough. E.g. we have to create a large training set to use neural network. It is unlikely that after addition of a new ideal gesture the training set could be quickly expanded for it. In contrast, with k-means algorithm it is hoped that it wouldn't require too many elements in the training set. The method of k nearest neighbors, as mentioned, require additional memory cost. And Bayesian method [1] is designed for special feature sets. Therefore, we have selected k-means algorithm for training.

Typically, the center of mass of a few vectors is calculated as sum of these vectors divided by the number of points. The problem of feature sets of non-fixed cardinality is that it is difficult to determine sum of feature sets and division by a scalar, so that geometric meaning of feature set was kept and training gave results.

For construction of center of mass of feature sets of non-fixed cardinality the following properties of center of mass were used:

1) Center of mass of one element is equal to this element.
2) Let M_n be center of mass of a set of n elements. We add element *k* to the set.

Let $d = dist(M_n, k)$ be the distance between the center of mass and the added element.

Let M_{n+1} be the new center of mass.

$$dist(M_{n+1}, M_n) = \frac{d}{n+1} = d0_M$$

$$dist(k, M_n) = \frac{d \cdot n}{n+1} = d0_k$$

We construct a point, that is located at distance $d0_M$ from point M_n, and at a distance $d0_k$ from point *k* (Property 2). In general, such point may not exist in a metric space. Then we construct a point, the ratio of the distances to which from M_n and *k* is the closest to desired ratio. Several points may satisfy this requirement, it is important that in this case the algorithm should return at least one such point.

Suppose we have two basic feature sets at a distance *d* from each other, and we have to build a basic feature set at distance $\frac{d}{n}$ from the first set, and $\frac{d \cdot (n-1)}{n}$ from the second. For calculating distance between two basic feature sets we consider pairs of cells that are at the minimum distance from each other. For each cell from the first basic feature set we find the nearest cell from the second base set, and for each cell of the second basic set nearest cell from the first basic set is found. We consider obtained pairs of cells as endpoints to construct the required set on a plane. Then we get a point, that divides the segment in a given ratio of $\frac{1}{n}$ from the first cell, if the first cell is referred to the first feature. Otherwise ratio is equal to $\frac{(n-1)}{n}$. The resulting points are interpreted as festures, and they can have non-integer coordinates. Basic feature set is consisted of points, distance to which from two given sets is equal to required value.

Center of mass of a set of *n* basic feature sets is constructed by induction. One basic feature set is equal to its center of mass. Let the center of mass of the set of *k* basic feature sets is already constructed. We add another basic feature set and calculate the distance *d* between it and the center of mass. By above-described method we construct new center of mass so that the distance from it to old center of mass is equal to $\frac{d}{k+1}$, and the distance from new center mass to the added element is equal to $\frac{d \cdot k}{k+1}$.

During the training we should change not only the center of mass, but the maximum distance from the center of mass of the class to a feature sets for the gestures of a given class. It is necessary to correct the maximum distance so that the sphere $B_n(M_n, D_n)$ would belong to the sphere B_{n+1} with center M_{n+1} and radius D_{n+1}. Added feature set should belong to the new sphere. Thus $D_{n+1} \geq \frac{d \cdot n}{n+1}$. Let $b_0 \in B_n$, B_n is feature set. By triangle inequality

$$dist(b_0, M_{n+1}) \leq dist(b_0, M_n) + dist(M_n, M_{n+1})$$

We recalculate maximum distance based on above inequalities

$$D_{n+1} = max(\frac{d \cdot n}{n+1}, D_n + \frac{d}{n+1})$$

VIII. EXPERIMENTS

For testing we selected ten objects that are used in QReal diagrams. The set of ideal multistroke gestures was generated by graphical representation of these objects (pic 1 – pic 10). Six people created training and test sets to test the effectiveness of different algorithms. There are1545 gestures in test set and 100 gestures in training set.

Number of cells in circumscribed about the user gesture rectangle that is used to construct the basic feature set is equal to 81. Recognition results are shown in the table 1.

The dependence of the recognition from number of cells in the rectangle (similar to image resolution) for combination of two algorithms is shown in the table 2. Note that after some threshold recognition accuracy does not increase with the size of the rectangle, but the speed drops due to the complexity of the algorithm. So we selected rectangle sizes equal to 81.

Percentage of recognized user gestures is very different for different objects. Recognition depends on similarity between different ideal gestures: if two ideal gestures are similar enough, user gesture that is corresponding to one ideal gesture can be confused with another. Algorithm based on matrix of number of cells in rectangle has a low general percentage of recognition, but some gestures that are poorly recognized by other algorithms, are well recognized by it. It was decided to combine feature vector from this algorithm with the feature vector that gives the best summary recognizability. Coefficients for linear combination of distances were chosen empirically, they are equal to 0.2 and 0.8 for algorithm based on matrix of number of cells and algorithm based on matrix of distance to the basic feature set respectively. Results of gesture recognition by combination of feature vectors are given in the last two rows of table 1.

Increasing of the percentage of recognized gestures depends not only on the training set, but the method of constructing feature vector. For example, feature vector based on matrix of number of cells in the zone shows improvement in recognition of 19% after training. While feature vector based on matrix of number of cells in rectangle shows recognition improvement of only 6%, but it shows better recognition percentage before training.

IX. CONCLUSION

Several algorithms of multistroke mouse gestures recognition have been tested. Necessary condition for all these algorithms is recognition independence of the number and order of gesture strokes and mouse movement direction.

After testing of algorithms without training we concluded that recognition without training doesn't satisfy the user. k-mean training algorithm has been applied for feature vectors of fixed dimension and was extended to the feature set of non-fixed cardinality. However, the initial recognition threshold and speed of training allows developers to provide the ability of training application online. Retraining after insignificant expansion of ideal gesture set is unnecessary. Initial detection threshold is the percentage of gestures recognized without training, the speed of training is the ratio of gestures recognized with training to gestures recognized without training. At the moment, the best recognition percentage after training is 91% on the test set of 1545 user gestures.

REFERENCES

[1] Cesar F. Pimentel, Manuel J. da Fonseca, Joaquim A. Jorge, «Experimental evaluation of a trainable scribble recognizer for calligraphic interface» // Graphics recognition: algorithms and applications, eds. Dorothea Blostein, Young-Bin Kwon, Ontario, Canada, 2001, pp. 81-85

[2] Don Willems, Ralph Niels, Marcel van Gerven, Louis Vuurpijl, «Iconic and multi-stroke gestures recognition» // Pattern recognition, Vol 42 Issue 12, pp. 3303-3312, December, 2009

[3] Kanungo, T.; Mount, D. M.; Netanyahu, N. S.; Piatko, C. D.; Silverman, R.; Wu, A. Y., «An efficient k-means clustering algorithm: Analysis and implementation». IEEE Trans. Pattern Analysis and Machine Intelligence Vol. 24 Isuue 7, pp 881–892, July, 2002

[4] Kelly, S., Tolvanen, J.-P., «Visual domain-specific modeling: benefits and experiences of using metaCASE tools», DSM Forum, URL http://www.dsmforum.org/papers/Visual_domain-specific_modelling.pdf

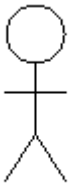
[5] Timofey Bryksin, Yuri Litvinov, Valentin Onossovski, Andrey N. Terekhov, «Ubiq Mobile + QReal - a technology for development of distributed mobile services», unpublished

[6] Timofey Bryksin, Yuri Litvinov, «QReal metaCase technology overview», unpublished

Pic 1: AcceptEventAction

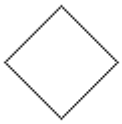
Pic 2: AcceptTimeEventAction

Pic 3: ActivityPartition

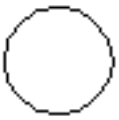


Pic 4: Actor

Pic 5: Comment



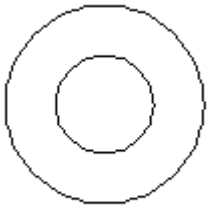
Pic 6: Decision Node



Pic 7: InitialNode

Pic 8: InputPin

Pic 9: SendSignalAction



Pic 10: FinalState

Algorithms	Picture										All gestures
	1	2	3	4	5	6	7	8	9	10	
User recognition	160	202	177	84	152	187	210	161	148	64	1545
Hausdorff distance	143	179	165	34	21	174	205	6	76	57	1060
Hausdorff distance + training	140	171	167	61	125	181	200	83	113	60	1301
Matrix of distance to basic feature set	141	176	171	39	21	178	204	19	80	58	1087
Matrix of distance + training	141	197	168	81	123	181	194	97	129	60	1371
Cells number at rectangle	45	144	169	10	85	51	193	8	82	33	820
Cells number at rectangle + training	79	112	171	28	112	99	164	70	41	39	915
Cells number at zone	112	169	166	5	15	61	205	7	44	25	809
Cells number at zone + training	122	114	174	44	133	142	185	69	62	49	1094
Combination of feature vectors	144	194	177	20	92	161	205	17	86	59	1155
Combination of feature vectors + training	140	197	175	74	149	180	195	126	115	59	1410

Table 1: Recognition results

Rectangle width and height, cell	32	40	50	60	70	81	90
Recognition, %	60	69	72	73	74	75	74

Table 2: Dependence of recognition on rectangle size

Novel heuristics for deconvolution applied to picture deblurring

Evgeniya Olenuk, Maxim Gromov
Faculty of Radiophysics
Tomsk State University
Tomsk, Russia
Email: evgeniya_ok@mail.ru, gromov@sibmail.com

***Abstract*—In this paper we describe an implementation of a method for blurred pictures restoring. We consider uniform, linear photo-camera shift (movement) while picture is taken. In this case, a picture, formed on a chip of a camera, is not static, but moving along some straight line, and as a result photo is blurred. Blurred picture can be seen as a convolution of two functions – one describes initial picture, another is point-spread (or blurring) function (PSF) [1]–[3]. "Deblurred" picture can be found as a deconvolution of the given picture with known PSF [1], [2]. To do this it is convenient to use Fourier transform and apply the convolution theorem. But it leads to division by 0, and thus the method is unstable. We suggest two heuristics to avoid division by 0 and compare them with the known one [2].**

***Index Terms*—Digital picture processing, blurred picture, Fourier transform.**

I. Introduction

Rapid development of computers brings lots of new possibilities into the world of picture processing. Some operations become easier with a computer – like colour correction, or appliqué creation etc – some become possible by themselves. One of such operation, which has no "manual" implementation, is picture deblurring [1]–[3].

In this paper we tell about our program which restores blurred pictures, taken by a digital camera. We use well-known method of deconvolution of two signals [1], [2]. The idea of the method is to describe blurring as a convolution of two functions and then to use Fourier transform to solve corresponding equation. However, this equation does not have a solution in general, since at some points it requires division by 0. To overtake this hardship there are several approaches.

One is known as Wiener deconvolution [4]. It relies on the fact, that in practice there is no measurement of a signal which can be done without noise. So the noise-to-signal ratio is never 0 and being added to the denominator of the deconvolution fraction it helps to avoid division by 0.

Another one is suggested by the authors of [2] and actually is application of the regularizing theory by Tikhonov [5]. They also modify the denominator of the deconvolution fraction in such a way, that it never turns 0. The idea comes from the fact, that we can tolerate the solution to be not exact but approximate.

Also we should mention Richardson-Lucy deconvolution [6]. It does not use Fourier transformation and being an iterative method it is considered to be very slow. Its implementation is used, for example, in ASTRA IMAGE software [7].

All mentioned methods assume the point-spread (or blurring) function (PSF) to be known. However there are bunch of works, devoted to automatic PSF reconstruction [3], [8], [9].

In our work we assume PSF to be known and to be one of the simplest type: uniform, linear shift of a camera. We use Fourier transform based deconvolution and suggest two new heuristics to avoid division by 0, which prove to work well according to our experiments, and which, we believer, are a bit simpler then Wiener or Tikhonov deconvolution. The idea is to modify denominator of the deconvolution fraction only when there is a need in division by 0. Then we replace denominator with 1, so there is no any division (first heuristics) or with the value which was used in divison at previous point (second heuristics).

The rest paper is structured as follows. In section II brief description of the method is given. Section III tells about the application itself. In section IV we show some experimental results, and section V concludes the paper.

II. Method description

A. Picture and its blurring

A picture may be defined as a function $u(x, y)$, where x and y are coordinates on a plane of the camera sensor [4]. The value of u at a point (x, y) is intensity of the light beam which has reached this point.

If the camera or the picturing object is moving while shutter is open, then a light point, formed on the sensor by the camera lens, will travel along some line and as a resulting in a blurred picture (Figure 1).

In our work we assume, that the camera (or object) motion is straight-line and uniform (without acceleration). In this case the travelling line of light points will be rectilinear segment of a length L, and the light intensity of the point will be uniformly distributed along this line. Without loss of generality we can assume, that the movement occurs along the axis x. Then intensity of light $u(x, y)$, measured by the sensor at the point (x, y), can be counted by the following formula $u(x, y) = \frac{1}{L}v(x, y) + \frac{1}{L}v(x + \Delta x, y) + \frac{1}{L}v(x + 2\Delta x, y) +$

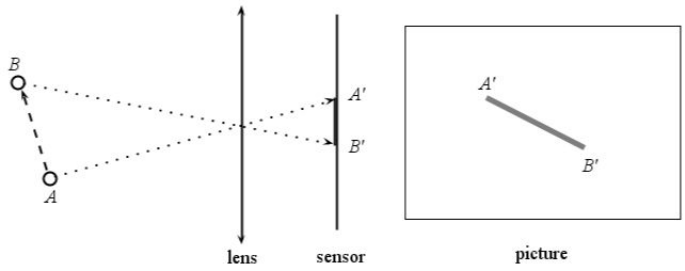


Figure 1. Scheme of formation blurred picture

$\ldots + \frac{1}{L}v(x+L,y)$, or in integral form

$$u(x,y) = \int_{-\infty}^{\infty} k(x-\xi)v(\xi,y)d\xi, \tag{1}$$

where $v(x,y)$ – picture which would be got, if there were no movement (and blurring),

$$k(x) = \begin{cases} \frac{1}{L}, & \text{if } x \leqslant L, \\ 0, & \text{otherwise} \end{cases}$$

is blurring function.

The expression (1) is a convolution-like equation for function $v(x,y)$, which is a original unblurred picture. Applying Fourier transform to the both sides of the equation (1) and keeping in mind convolution theorem [10] we shall get algebraic equation for Fourier image $V(\omega,y)$ of the function $v(x,y)$: $U(\omega,y) = K(\omega)V(\omega,y)$. Unfortunately, $|K(\omega)|$ turns 0 (or, with no difference, very small) for some values of ω and in this points solution could not be found as

$$V(\omega,y) = \frac{U(\omega,y)}{K(\omega)} \equiv \frac{U(\omega,y)K^*(\omega)}{|K(\omega)|^2}. \tag{2}$$

B. Different approaches to avoid division by 0

There are several approaches to avoid division on 0 in expression (2). The historically first approach should be considered deconvolution, based on Wiener's filter [**?**]:

$$V(\omega,y) = \frac{U(\omega,y)K^*(\omega)}{|K(\omega)|^2 + \mathrm{NSR}(\omega)},$$

where $\mathrm{NSR}(\omega) > 0$ is noise-to-signal ratio, and since it is hard to have a measurement without noise, this function in practice is never 0. In most application it can be replaced with some a priory known constant.

In [2] it is suggested to find solution in the form

$$V(\omega,y) = \frac{U(\omega,y)K^*(\omega)}{|K(\omega)|^2 + \alpha\omega^{2p}},$$

where α and p are heuristics parameters. This formula comes from the regularizing theory by Tikhonov [5]. It is easy to see, that in certain cases Tikhonov deconvolution turns into Wiener deconvolution, for example, when $p = 0$ and $\alpha = \mathrm{NSR} = \mathrm{const}$.

We use a little bit different approach. We suggest to keep formula (2) intact while there is no division by 0 and only in cases of such ω, that $|K(\omega)|^2 < \varepsilon$ replace $|K(\omega)|^2$ with 1 (then there is no division), or with the value which was used obtained for previous ω. Here ε is a priory specified (by user) threshold.

C. Digital case

Since all pictures we deal with are taken by digital camera, function $u(x,y)$ (as well as $v(x,y)$ and $k(x)$) and its arguments x and y will take discrete values within certain limits. Usually, for 24-bit `.bmp` files, which we shall use for our application, value of intensity of a channel (red, green or blue) lies within 0 and 255, 0 stands for the lowest intensity (no light), 255 – the highest. Intensity, which is less then the lowest is considered to be 0, and which is more then the highest – 255.

Values are measured equidistantly all over camera's sensor, which is usually rectangular. Points of the measurement are called pixels. Upper left pixel has coordinates $(0,0)$, next right to it – $(1,0)$ etc; next bottom to the pixel $(0,0)$ is pixel $(0,1)$ etc. Thus, digital picture is a matrix $\mathbf{u}$, where $\mathbf{u}(x,y)$ is intensity, measured at pixel (x,y), $x \in \{0,1,\ldots m-1\}$, $y \in \{0,1,\ldots n-1\}$, m – width of the picture in pixels, n – its height. Note, that we consider first index of the matrix to be its column, and second – row.

In the digital case, the integral in (1) should be replaced by the sum:

$$\mathbf{u}(x,y) = \sum_{\xi=0}^{m-1} \mathbf{k}(x-\xi)\mathbf{v}(\xi,y), \tag{3}$$

and Fourier transform becomes the discrete Fourier transform, when we try to find a solution for (3):

$$\mathbf{V}(\omega,y) = \frac{\mathbf{U}(\omega,y)}{\mathbf{K}(\omega)}. \tag{4}$$

Here $\mathbf{V}$ – is the Fourier image of the unblurred digital picture $\mathbf{v}$, $\mathbf{U}$ – is the Fourier image of the blurred digital picture $\mathbf{u}$, taken by the camera, $\mathbf{K}$ – is the Fourier image of the blurring vector $\mathbf{k}$. Again, for some ω $|\mathbf{K}(\omega)|^2$ may turn 0 or become small enough to make division in (4) invalid. If that happens, our program replaces $|\mathbf{K}(\omega)|^2$ with 1 or replaces $\mathbf{K}(\omega)$ with the value $\mathbf{K}(\omega-1)$, depending on what heuristics was chosen by a user.

We measure blurring distance L in pixels, which, we think, is the most convenient way. So, the value of L is integer and has the following meaning. Suppose $L = 5$. Then a light point, which in other circumstances would result as a pixel (x,y) with intensity $\mathbf{v}(x,y)$, "spreads" its intensity among L pixels (x,y), $(x+1,y)$, $(x+2,y)$, $\ldots(x+4,y)$ and each gets $1/L = 1/5$ part of the original intensity $\mathbf{v}(x,y)$, and thus $\mathbf{k} = \langle \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 0 \ldots, 0 \rangle$.

III. Method implementation

For implementation of the method, described in section II, we have chosen Java and NetBeans [11] as IDE. For the discrete Fourier transform we decided to use The Apache Software Foundation implementation [12] of Fast Fourier Transform algorithm [10]. This fact implies some restrictions

for the width m of a picture: it should be of an integral extent of 2. We decided m to be $2^{10} = 1024$.

The application works with files of 24-bit `.bmp` pictures of width 1024 pixels. Due to our assumption, that blurring is happening along axis x, each line of the picture (the row in terms of matrix) is processed separately (and the line has three independent channels: red, blue and green).

Interface of the program you can see in Figure 2. On the left

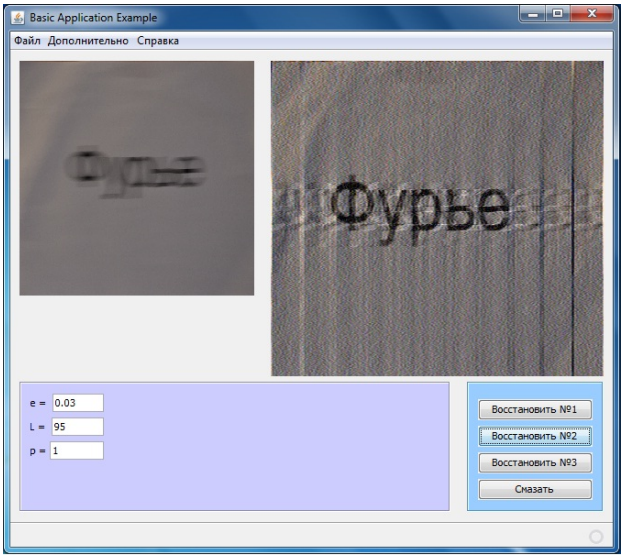


Figure 2. Interface

part of the window there is original picture and on the right – processed. User can chose either to restore the original picture (buttons "Restore No. 1", "Restore No. 2", "Restore No. 3") or blur it (the button "Blur").

Value of e is the threshold for our heuristics. If the value of $|\mathbf{K}(\omega)|^2$ is less then e, program replaces $|\mathbf{K}(\omega)|^2$ with 1 (i.e. there is no division for this ω in (4)), when the first heuristics is chosen (button "Restore No. 1"), or replaces $\mathbf{K}(\omega)$ with the value $\mathbf{K}(\omega - 1)$, when the second heuristics is chosen (button "Restore No. 2"). In case the user presses button "Restore No. 3" Tikhonov deconvolution will be used with α and p taken from edit-fields e and p correspondingly.

Value of L stands for the blurring distance. The user can specify it by hand or by clicking mouse on the original picture.

IV. Some experiments with the program

Here we present some examples of pictures, restored by our program. First we use artificially blurred picture (Figure 3 b)), blurring distance L is 80 pixels. Restored picture is shown in Figure 4. There we used first heuristics: replacement of values of $|\mathbf{K}(\omega)|^2 < e$ with 1, where $e = 0.01$.

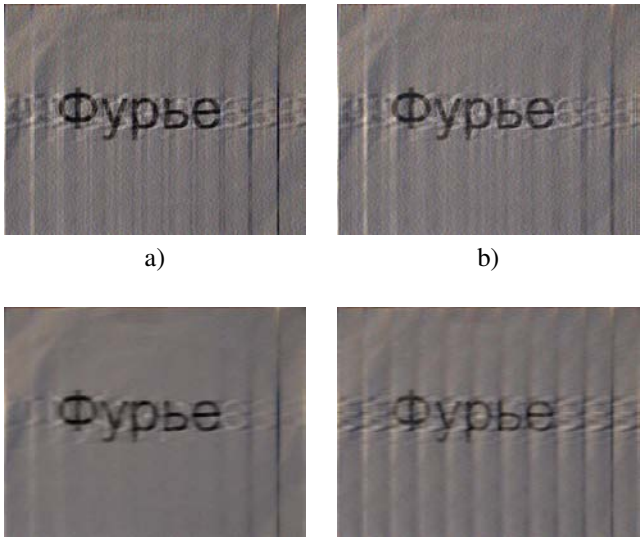
Then we took a picture of a sheet of paper with Figure 3a while moving the camera (see Figure 5). Result of restore is shown in Figure 6.

a) b)

Figure 3. Original picture (a) and blurred by Photoshop [13] (b)

Figure 4. Restored picture from Fig. 3b, $e = 0.01$, $L = 80$

Figure 5. Really blurred picture



a) b)

c) d)

Figure 6. Restored picture from Fig. 5: a) "invalid" values of $\mathbf{K}(\omega)$ are replaced by 1, $e = 0.03$, $L = 95$; b) "invalid" values of $\mathbf{K}(\omega)$ are replaced by $\mathbf{K}(\omega - 1)$, $e = 0.04$, $L = 95$; c) using Tikhonov's method with $\alpha = 0.01$ and $p = 0$; d) using Tikhonov's method with $\alpha = 10^{-6}$ and $p = 1$

V. Conclusion

In this paper we have presented our implementation of the picture deblurring method, based on deconvolution. Blurring

process in most cases is a process with loss of data: at some points value of Fourier image $K(\omega)$ of PSF $k(x)$ becomes 0, causing loss of data. This means, that equation (1) has no solution for $v(x, y)$ (deblurred picture). To get some function, close enough to the solution of (1) authors of [2] suggested to use Tikhonov deconvolution, replacing $1/K(\omega)$ with $K^*(\omega)/(|K(\omega)|^2 + \alpha\omega^{2p})$, where α and p are parameters of their heuristics. Another well-known approach is to use Wiener's deconvolution [4] Here we have developed our own heuristics: replace $K(\omega)$ with 1 or replace $K(\omega)$ with the value from previous point whenever $|K(\omega)|^2$ becomes less then some predefined by a user threshold. Experiments have shown, that our approach is capable to work with artificially created blurred pictures as well as with blurred pictures taken by real camera.

REFERENCES

[1] V.S. Sizikov *Stable methods for the measurement results processing*, Tutorial, Saint-Petersburg: "SpetzLit", 1999 (in Russian).

[2] A.G. Yagola and N.A. Koshev *Restoring of blurred and defocused pictures*, Computational methods and programming, Moscow, Russia: Moscow University publishing house, Vol. 9, 2008, Pp.207-212 (in Russian).

[3] N. Joshi, S.B. Kang, C.L. Zitnick and R. Szeliski *Image deblurring using inertial measurement sensors*, ACM Transactions on Graphics (TOG) – Proceedings of ACM SIGGRAPH 2010, Volume 29 Issue 4, July 2010 ACM New York, NY, USA (available at http://research.microsoft.com/en-us/um/redmond/groups/ivm/imudeblurring/).

[4] R. Gonzalez and R. Woods *Digital Image Processing*, Moscow, Russia: Techosphere, 2005 (in Russian).

[5] A.N. Tikhonov and V.Ya. Arsenyev *Incorrect problems solving methods*, Moscow: Nauka, 1979 (in Russian).

[6] URL: http://en.wikipedia.org/wiki/Richardson-Lucy_deconvolution

[7] URL: http://www.phasespace.com.au/decon_ex.htm

[8] URL: http://en.wikipedia.org/wiki/Blind_deconvolution

[9] Y.-W. Tai, H. Du, M.S. Brown and S. Lin *Image/video deblurring using a hybrid camera*, In Computer Vision and Pattern Recognition, 2008.

[10] A.V. Aho, J.E. Hopcroft and J.D. Ullman *The Design and Analysis of Computer Algorithms*, Moscow, Russia: Mir, 1979 (in Russian).

[11] URL: http://netbeans.org/

[12] URL: http://commons.apache.org/math/

[13] URL: http://www.photoshop.com/

A Semiotic Approach to the Intelligent Chinese CALL System Development

Chuprina Svetlana
Department of Computer Science
Perm State National Research University
Perm, Russia
chuprinas@inbox.ru

Osotova Tatyana
Department of Computer Science
Perm State National Research University
Perm, Russia
hvostya@gmail.com

Abstract— **In this paper, we present a novel approach to development of intelligent Chinese Computer-Aided Language Learning (CALL) systems based on ontological engineering methods and a semiotic model of Chinese hieroglyphs. The new features and methods such as "teaching by doing" method, mnemonic novel method and others make the learning process easier and are involved due to above mentioned approach. A semiotic model of Chinese hieroglyph in terms of the first-order logic is described. We have developed the system OntoKit 2.0 based on the results of our full research.**

Chinese CALL system; semiotic model of Chinese hieroglyph; ontological engineering method; mnemonic novel method

I. INTRODUCTION

An integration of traditional and intelligent information technologies (IT) is one of mainstreams of modern software development and of CALL systems implementation in particular. Recently, the most implementations of traditional CALL systems for non-native learners are focused on automation of learning material development, language assessment and learner's training as usual. In our opinion there are not enough in order to get the best learning outcomes. Additionally it is necessary to introduce creative elements to improve learner's cognition skills, to give an opportunity to reveal new regulations and patterns of application domain via self-training (for example, via cognitive games).

This paper presents one of possible approaches to develop intelligent CALL systems as an adaptive self training system and to solve mentioned above problems. We describe the implementation of an intelligent Chinese writing CALL system OntoKit 2.0 environment (see [1, 2] for details). Its ontological knowledge base includes description of semiotic model of Chinese hieroglyph as well as descriptions of the other types of knowledge (tree structure of hieroglyph, syntactical rules of Chinese language, etc). OntoKit 2.0 has became adaptive to different hieroglyphic Asian languages' features due to ontological reengineering methods and universality of supported paradigmatic relations.

In a way we are considering the problem of automation of Chinese characters training as an application domain of integration of such different technologies as pattern recognition (including Artificial Neural Networks), semantic networks and ontological engineering to improve the intelligent capabilities of Chinese CALL systems. The using of techniques mentioned above let us to add to automated training systems the new features to evaluate Chinese characters learning from a simple repetition to the creative process [3]. Ontology based approach makes it possible not only to introduce the new features and methods such as "teaching by doing" method, "mnemonic novel" method, taking into account the historical and culturological aspects of hieroglyphs' evolution into learning process but also provides an original ability of CALL system to advance itself by extension of knowledge base without the necessity to change the source code [1].

The phonological and the phonetic aspects of Chinese language learning are not the issue of our research.

II. METHODS OF ACCELERATION OF CHINESE HIEROGLYPHS MEMORIZATION

"Indo-European languages are based on a finite alphabet" and "letters do not carry meaning unless they are strung together into words" while Chinese language is "made up of symbols that themselves embody meaning", and "the number of possible symbols or elements in these languages is arbitrarily large and can be considered infinite" [4]. And for those who have just started acquaintance with Chinese written language, it seems that there is an incredible variety of characters, and it is almost impossible to remember that.

As mentioned above many CALL systems and Chinese CALL systems in particular are simply a set of lessons with some characters that you just need to learn, and there is no explanation of hieroglyph's structure, which would greatly facilitate the task of the study of Chinese written language for learners.

Chinese language is classified as ideographic, that is, to a system of signs which are used to record the lexical meaning of linguistic units [5]. We support the idea that an image of most of hieroglyphs maps the form of an object (physical or abstract entity) or a group of objects of the reality. In time the images of many Chinese characters became more schematic and abstract. These were the so-called "ideograms", which eventually got the form of modern Chinese hieroglyphs, a lot of which lost its original expressive image during the process of evolution. For example, the character shown in Fig. 1 is perceived not like a simple sequence of strokes, but as a schematic representation of the tree also.

Figure 1. Chinese hieroglyph “Tree” (on the left) and picture of tree (on the right)

However, our imagination can extract the original visual image from the abstract scheme as before [3] if historical and culturological aspects are taken into account especially (see Fig. 2).

The graphics and semantics of Chinese hieroglyphics’ evaluation are considered as a common process. Not only complex Chinese character as a whole, but parts of it (graphemes) carry certain semantic load. Therefore, the meaning of a complex Chinese character depends on the meanings of its graphemes usually [3]. For example, Fig. 2 illustrates the image of Chinese character "Prosperity", which consists of the characters "Woman" and "Roof", so it is the symbol of prosperity for the Chinese people because it describes a situation when “a woman is at home” (under rooftop). And the character "Woman" itself depicts a sitting woman, which bended knees (this pose was accepted for sitting in the ancient China) and with folded hands in a sign of submission [5].

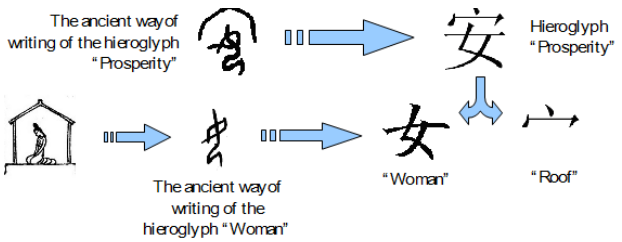
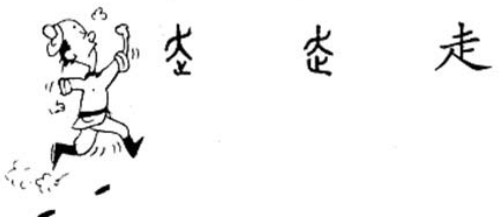


Figure 2. Historical and culturological aspects of hieroglyphs “Prosperity” and "Woman"

Taking account of historical and culturological aspects related to the evolution of the graphical representation of hieroglyphs during the process of Chinese characters learning, makes it possible not only to acquaint the learner with the history and culture of China and to feel the mentality of Chinese people better, but also to improve the process of Chinese characters learning itself [3].

However, it happens that it is not enough to learn only the appearance and modern spelling of some Chinese character to recognize its meaning from graphical representation [3]. Then for character learning it requires to create an additional stimulus in the form of visual metaphor to exert influence on learner’s memory [6] (see Fig. 3). Some kind of modification of the mnemonic novel method is used in our approach for this purpose: we take into account images of one and the same



hieroglyph from different historical epochs and some additional facts (such kind of knowledge is stored at the ontological base) [3].

Figure 3. An example of the way of graphic visual metaphor creation for Chinese character "to go" [7]

III. MODEL OF CHINESE HIEROGLYPH

In contrast to the Indo-European languages single Chinese character is considered as a whole word or a part of the word. We have constructed a formal model for the description of the semiotic structure of the Chinese character to solve problems of explanation of Chinese characters’ structure and its features to learners.

The first-order logic is used to formalize the model. However, there are all sorts of ambiguities by reason of incomplete knowledge, because the Chinese language is extensible. In addition, it should be noted there is some inconsistency between different sources about the nature of many Chinese characters’ structure. For example, even such obligatory element of any hieroglyph as a radical has different definitions. In one case the radical is defined via graphemes, but in another case it can be defined via other graphical elements. Thus it is not enough to describe the formal model of Chinese hieroglyph in whole by the apparatus of first-order logic, and we use an apparatus of nonmonotonic logic, the default logic of Reiter (see [8] for details).

The formal model of Chinese hieroglyph in the OntoKit 2.0 is a triple

$$V=\{S,T,K\}, \quad (1)$$

where S — a simbol of Chinese hieroglyph itself (symbol is considered as a part of Frege semiotic triangle (see [9,10] for details) showed on Fig. 4), T — translation of the Chinese hieroglyph into Russian language, K — knowledge about the hieroglyph, concerning both its structural component (image syntax) and semantics, including historical and culturological aspects [11].

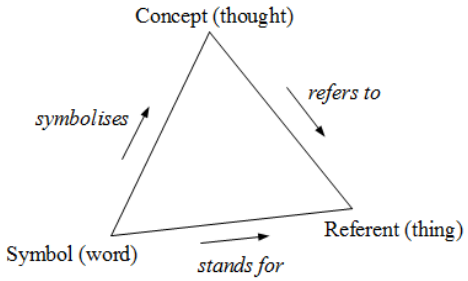


Figure 4. Semiotic Frege Triangle

Let us make some comments on terminology.

Any hieroglyphic character both complex and primitive consists of some number of standard graphic elements, named *strokes* [12]. The set of strokes is restricted (see, for example, Fig. 5). Graphical indication of the stroke is that it consists of one persistent line (see [13] for details).

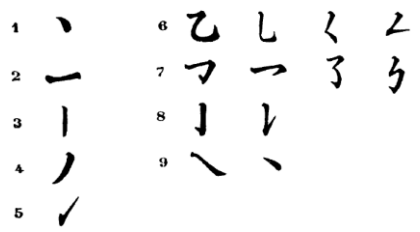


Figure 5. Base strokes of the Chinese hieroglyphics according to [14]

It seems to us that there are no requirements to organize the semantic search based on strokes only.

Additionally there are 24 basic graphical structures (some of them look as a stroke) so called "features" picked out especially for Chinese printed text (Fig. 6). Some dictionaries, for example [15], let conduct the search based on features mentioned above.

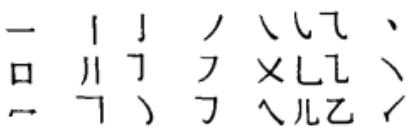


Figure 6. 24 typical features, which are marked out for Chinese printed text [15]

However, representation of hieroglyphic symbol only as a combination of strokes is not enough to understand its meaning. "Meaning" is considered here as a "referent" from the Frege triangle (see [9, 10] for details) showed on Fig. 4. That's why more complex graphical constructions such as graphemes and radicals are required.

Combinations of strokes with fixed lexical meaning are called *graphemes* (see [12] for details). As the result of our review of the different interpretations of the structure of the Chinese characters information resources we have considered that not all graphemes have a self-contained lexical meaning. On our point of view the inner "context" within the character's image has the significant impact on the recognizing of the grapheme's meaning.

Hieroglyph's structure mathematically represents an ordered tree. Sharapov J.A. in [16] (the author of paper [16] is OntoKit development group member as well as the authors of this paper) points out that generally graphemes may be represented as the leaves of the latter tree. The graphemes' order in the hierarchical structure of hieroglyph is an important part to comprehend its meaning. That's why the ontological knowledge base includes semantic description of hieroglyph's structure. This description is handled by special component responsible to break down (to decompose) Chinese characters to its parts automatically. Description of the model of Chinese character represented as an ordered tree, the method of hieroglyph's decomposition to graphemes and the adequate internal view construction for Chinese characters is beyond the scope of this paper (see [16] for details).

Primitive Chinese character is hieroglyphic symbol consisting of a single grapheme.

Complex Chinese character is hieroglyphic symbol consisting of more than one grapheme.

The *radical* is a semantic component of hieroglyph and hints at the meaning of the character. Within the environment of OntoKit 2.0 mentioned above a radical represents the belonging of any Chinese hieroglyph to the concrete semantic category of an appropriate domain specific ontology (see [3] for details). For example, consider a fragment of domain ontology shown in Fig. 7. We can see that the hieroglyph 森 has the radical 木 "tree" (it belongs to the category "Nature" of domain ontology). It signifies that translation of hieroglyph 森 is related to tree (the meaning of this Chinese character is "thick forest").

Figure 7. Fragment of domain ontology from OntoKit 2.0

According to [12] the radical is either some stroke (for example, radicals "one", "vertical", "tilting to the left") with no fixed meaning, or a grapheme, or symbol, consisting of 2-3 graphemes (for example, the radicals "to see", "chamois leather", "hemp", "turtle", "flute") as it is showed on Fig. 8. Earlier there were 214 radicals. A total list of radicals has been revised by Chinese linguists recently. The graphic elements, which were used in a limited number of hieroglyphs or rare, have been excluded. The modern list of radicals includes about

190 radicals (different versions of some radicals are taken into account also) [17]. There are a lot of dictionaries based on the radicals search.

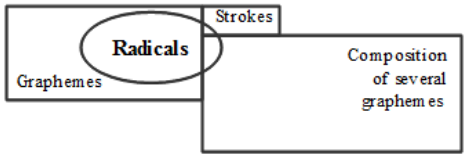


Figure 8. Illustration of corelation between set of Chinese characters' radicals and sets of other graphical elements (from the structure of the image point of view)

The model of Chinese hieroglyph in the language of first-order predicate is presented below (note that we give mnemonic names to predicates for convenience despite the fact that it is accepted to denote predicates by single symbols of the Roman alphabet):

"W" is a set of concepts.

"w" is some concept, $w \in W$.

"X" is a set of graphic elements derived from the strokes that can be called symbols in a semiotic perspective, that is, graphemes, including the strokes with independent meaning, and hieroglyphs

"x" is an element of the set X; $x \in X$.

S(x) asserts that x is a hieroglyph, $x \in X$.

Simp(x) asserts that x is a primitive hieroglyph, $x \in X$.

Comp(x) asserts that x is a complex hieroglyph, $x \in X$.

G(x) asserts that x is a grapheme, $x \in X$.

Key(x) asserts that x is a radical , $x \in X$.

BeKey(x,y) asserts that x is a radical to y ($x, y \in X$, where $Key(x) \wedge S(y)$ is correct).

Mean(x,w) asserts that graphical element x have meaning w, $x \in X$, $w \in W$.

P(x,y) asserts that x is a part of y; $x, y \in X$.

Eq(x,y) asserts that x is y; $x, y \in X$ (that is graphical elements x and y are equal by inscription).

EqW(x,y) asserts that x is y, $x, y \in W$.

$Op_{modif} = \{VC, HC, NTC\}$ is an operation of distortion, where *VC* is a vertical compression, *HC* is a horizontal compression, *NTC* is a nonformalizable transformation (it is usually associated with some fundamental simplification of the structure, and most of these transformations can be specified by the table).

$modif \in \{Op_{modif}, Op_{modif}(modif')\}$, где $modif' \in \{Op_{modif}, modif\}$.

1. Common statements:

$$(\forall x)(P(x,x)), \tag{2}$$

$$(\forall x)(\forall y)(Eq(x,y) \rightarrow Eq(y,x)), \tag{3}$$

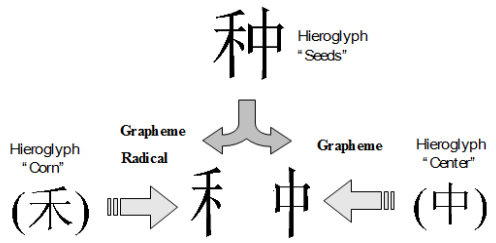
$$(\forall x)(\forall y)(Eq(x,y) \rightarrow (P(x,y) \wedge P(y,x))), \tag{4}$$

$$(\forall x)(\forall y)(\forall z)((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)). \tag{5}$$

2. Grapheme is a part of hieroglyph (Fig. 9)

$$(\forall x)(G(x) \rightarrow (\exists y)(S(y) \wedge P(x,y))). \tag{6}$$

Figure 9. Illustration of the construction features of Chinese characters



3. A Chinese hieroglyph consists of one grapheme at least (Fig. 9):

$$(\forall x)(S(x) \rightarrow (\exists y)(G(y) \wedge P(y,x))). \tag{7}$$

4. A Chinese hieroglyph is primitive if and only if it consists of a single grapheme (Fig. 10):

$$(\forall x)(Simp(x) \sim (S(x) \wedge (\forall y)((G(y) \wedge P(y,x)) \rightarrow Eq(x,y)))), \tag{8}$$

where "$\sim$" is a symbol of biconditional operation.

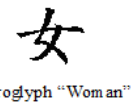


Figure 10. Example of primitive hieroglyph

5. A Chinese hieroglyph is complex if and only if it consists of more than one grapheme (Fig. 11):

$$(\forall x)(\,Comp(x)\sim(\,S(x)\wedge (\forall y)((G(y)\wedge P(y,x))\rightarrow \overline{Eq(x,y)}))\,), \quad (9)$$

where "$\sim$" is a symbol of biconditional operation (this statement does not exclude the situation when the complex character consists of several identical graphemes).

Figure 11. Example of complex hieroglyphs

6. A Chinese hieroglyph is either primitive or complex:

$$(\forall x)(S(x)\rightarrow (Simp(x)\oplus Comp(x))), \quad (10)$$

where "$\oplus$" is exclusive OR operation.

7. A radical is a part of hieroglyph (Fig. 9):.

$$(\forall x)(\forall y)(BeKey(x,y)\rightarrow(S(y)\wedge Key(x)\wedge P(x,y))), \quad (11)$$

8. There is a radical in any Chinese hieroglyph and it is just one (Fig. 9):

$$(\forall x)(\,S(x)\rightarrow (\exists y)(\,BeKey(y,x)\wedge (\forall z)(BeKey(z,x)\rightarrow Eq(z,y)))). \quad (12)$$

9. One and the same radical in the context of different characters can have different meanings [12]:

$$(\exists x_1)(\exists x_2)(\exists y_1)(\exists y_2)((\,BeKey(x_1,y_1)\wedge BeKey(x_2,y_2)\wedge Eq(x_1,x_2)\wedge \overline{Eq(y_1,y_2)})\wedge (\exists w_1)(\exists w_2)(\,Mean(x_1,w_1)\wedge Mean(x_2,w_2)\wedge \overline{EqW(w_1,w_2)})). \quad (13)$$

An example of such radicals is shown in Fig. 12: Chinese hieroglyphs "City" and "Hill" in the function of radical have the same spelling. These radicals can be discerned in the character only by their position: "City" is used on the right, "Hill" - on the left.

10. There are radicals with the same meaning but different spelling:

$$(\exists x_1)(\exists x_2)(\exists y_1)(\exists y_2)((\,BeKey(x_1,y_1)\wedge BeKey(x_2,y_2)\wedge \overline{Eq(x_1,x_2)})\wedge (\exists w)(Mean(x_1,w)\wedge Mean(x_2,w))). \quad (14)$$

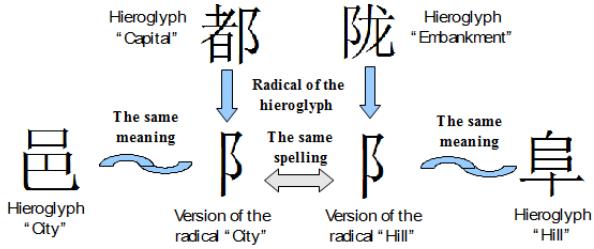


Figure 12. Radicals, which in the function of radical have the same spelling, but different meaning

Let us explain this statement with an example. In Fig. 13 there is shows the character "Fire", it is the radical of the character "Flame" (but in the character "Autumn" this character is only a grapheme). In addition, the character "Boil" has as the radical "Fire", although the spelling differs.

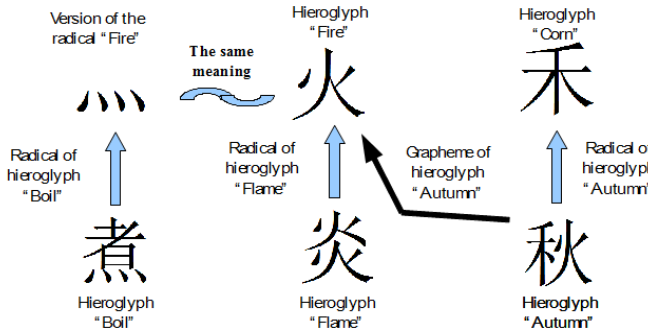


Figure 13. Illustration of the features of radicals of Chinese characters

The statements represented below are noted with default logic of Reiter.

11. A radical is a grapheme (Fig. 9), if it does not contradict with other knowledge represented in the system (in Fig. 8 it is reflected that this statement is not always true):

$$(\forall x)(Key(x)\rightarrow(\exists y)(G(y)\wedge Eq(x,y))), \quad (15)$$

$$\frac{Key(x):M(G(y)\wedge Eq(x,y))}{G(y)\wedge Eq(x,y)}. \quad (16)$$

12. Primitive hieroglyph associates with a grapheme with certain distortion (Fig. 9), if it does not contradict other knowledge represented in the system (some

primitive characters are presented by grapheme without distortion):

$$(\forall x)(\exists y)(\,Simp(x)\rightarrow (Eq(x, modif(y))\wedge G(y))\,), \quad (17)$$

$$\frac{Simp(x): M(Eq(x, modif(y))\wedge G(y))}{Eq(x, modif(y))\wedge G(y)}. \quad (18)$$

We use the model described above as a basis of the development of our CALL system OntoKit 2.0 and so called "teaching by doing" method in particular. Due to this method a learner has the facilities not only to decompose Chinese character to graphemes in order to know its meaning but to compose a new hieroglyph from different graphical constructions to "design" a new meaning based on the meanings of its source parts also [2]. It brings elements of creative work to the learning process and also provides an ability of CALL system to advance itself by extension of knowledge base without the necessity to change its source code [1].

A general scheme of decomposition process is illustrated in Fig. 14.

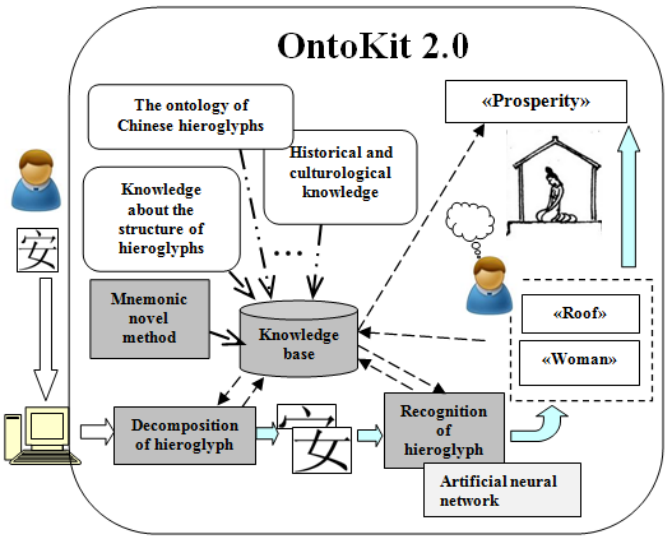


Figure 14. A general scheme of automation of Chinese characters learning process [18]

There is a scheme of hieroglyph's image analysis in Fig. 15. An image of hieroglyph (in bmp or jpg format, for example) is an input of this process. The system breaks down image to images of hieroglyph's components automatically. As the result of this process the internal view of hieroglyph's components description in ordered tree form is generated due to recognition based on the convolutional neural network using [19]. The system uses the results of recognition to help someone learn the Chinese hieroglyphs, memorize its spelling, or perform other learning-related activities via cognitive games.

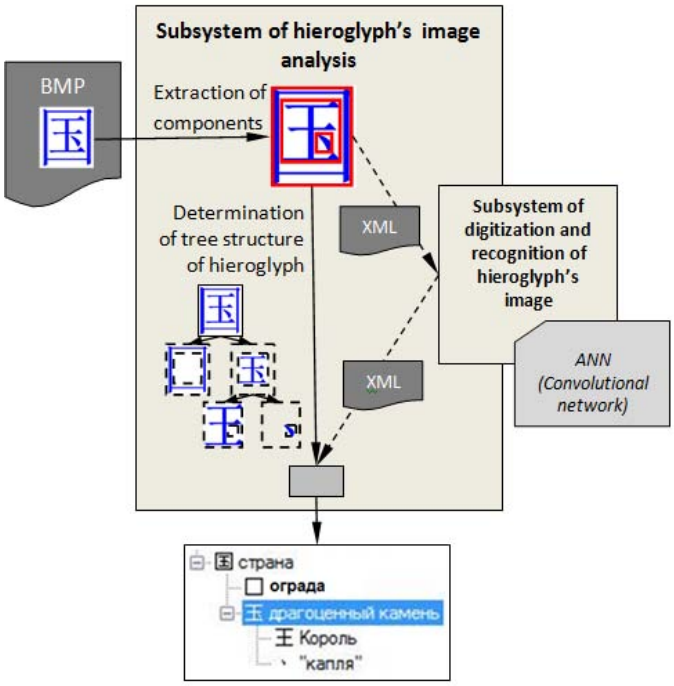


Figure 15. OntoKit 2.0: a scheme of hieroglyph's image analysis

CONCLUSION

This article focuses on the problems of automation of Chinese characters training and development of an extensible Chinese CALL system with new intelligent facilities based on the original ontological approach and semiotic model of hieroglyph.

The semiotic model of Chinese character in terms of the first-order logic and non-monotonic default logic of Reiter is represented.

We have implemented the research prototype of OntoKit 2.0 based on the proposed approach by using C# and C++ languages. So the viability of described approach have been proved. That's why we consider our results as full research.

REFERENCES

[1] Chuptina S.I., Sharapov J.A., Osotova T.V. "The ontology approach to creation of an automated adaptive system for teaching chinese characters" // Proceedings of XXXVI International Conference «Information technology in scince, sociology, economics и business» IT + SE'09 The Autemn session. M.: 2009, pp. 53-55. (rus)

[2] Osotova T.V. "The approach to Chinese character recognition at the CALL system OntoKit 2.0" // Proceedings of scintific academic conference "Modern problems of mathematic and its applied areas" Perm: Perm State University, 2010, pp. 114-118. (rus)

[3] Chuptina S.I., Sharapov J.A., Osotova T.V. "Automatisation of Chinese language training: historical and culturological approach" // Proceedings of scintific conference «Historical and Cultural Heritage and Information and Communication Technology: retention analysis». Perm: Perm State University, 2009, pp. 202-213. (rus)

[4] Jurgens H., Peitgen H.-O., Saupe D. “The Language of Fractals” // Scintific American.1990.№10.P. 36.

[5] Volkova O.N. “Culture-oriented linguistics of the first foreign language (chinese language). Module 1. The Chinese grammotology” Version 1.0 [Online resource]: online educational manual. URL: http://www.files.lib.sfu-kras.ru/ebibl/umkd/346/u_course_1.pdf (access date: 30.01.2012). (rus)

[6] Vurdov A.M. “Japanese language for pleasure. Kanji essays” Syktyvkar: Uki, 2006, p.528. (rus)

[7] Lusya V., Starostina S.P. “The Chinese-Russian educational dictionary”.– M.: AST: Vostok – Zapad, 2006, p.382 (rus)

[8] Ueno H., Koyama T., Okamoto T., Matsubi B., Isidzuka M.. “Knowledge representation and deployment” Trans. from jap. – M.: Mir, 1989, pp.199 – 207. (rus)

[9] Nesterov A.V. “On semantic, pragmatic, and dialectic triangles” // Automatic Documentation and Mathematical Linguistics, Vol. 43, №3. Allerton Press, 2009, pp. 9-14.

[10] Frege: “On Sense and Denotation” // UW Faculty Web Server. URL: http://faculty.washington.edu/smcohen/453/FregeDisplay.pdf (access date: 30.03.2012).

[11] Osotova T.V. “The role of semiotic description at automated Chinese language training” // Proceedings of IV International Online Scintific Students’ Conference «Students’ scintific forum» URL: http://www.rae.ru/forum2012/pdf/2721.pdf (access date: 26.03.2012) (rus)

[12] Kondrashevskii A.F. “Practical cource of the Chinese language. Hieroglyphic’s guide’” Part I. M. : PH «Muravey», 2000, p.152 (rus)

[13] Hieroglyph’s structure // Educational center «Sakura» URL: http://www.sakura-inyaz.ru/struktura_ieroglifov.html (access date: 15.12.2011). (rus)

[14] Wieger L. “Chinese Characters. Their origin, etymology, history, classification and signification. A thorough study from Chinese documents” – New York: Paragon book reprint corp. — P. 12.

[15] Panasyuk V.A., Suhanov V.F. “The Great Chinese-Russian dictionary” Vol. 1. M.: «Nauka», 1983, pp.10 – 12. (rus)

[16] Sharapov J.A. “The algorithm of inventive tasks solving at the task of Chinese characters’ structure description” // Proceedings of XXXVII International Conference «Information technology in scince, sociology, economics и business» IT + SE’10 The Spring session. M.: 2010, pp. 86-88. (rus)

[17] Internet-school of the Chinese language. Issue 4. The Chinese writing. Knowledges aboute the structure and spelling of hieroglyphs // Internet-school of the Chinese language URL: http://chinese-school.narod.ru/004.html (access date: 23.12.2011). (rus)

[18] Osotova T.V. “The semiotic approach to aftomation of Chinese character training” / The bulletin of Perm State University. Mathematics. Mechanics. Informatics. Issue 3(7). Perm, 2011, pp. 59-62. (rus)

[19] Quen-Zong Wu, Yann Le Cun, Larry D. Jackel, Bor-Shenn Jeng On-line recognition of limited-vocabulary Chinese character using multiple convolutional neural networks // Yann LeCun home page. URL: http://yann.lecun.com/exdb/publis/index.html (access date: 23.05.2009).

Scheduling Problem Solutions in Transport Enterprises

Andrey Orlov
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
lokaro.oa@gmail.com

Scientific Advisor: Prof. Sergey Avdoshin
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
savdoshin@hse.ru

***Abstract* – The classical transportation problem is a problem of optimal transportation plan of a homogeneous product of uniform items in the presence of homogeneous items of consumption on homogeneous vehicles with static data and the linear approach [1]. Development and application of optimal schemes of cargo flows can reduce the cost of transport and maximize profits. We should make a detailed transportation plan, which should take into account a number of limitations. Models for such tasks contain more 50,000 variables and constraints. The market hasn't practically any software, which satisfies all companies' needs. In this article we will consider the process of modeling in IBM ILOG CPLEX Optimizer. This software includes efficient algorithms for solving optimization problems.**

Keywords: optimization; modeling; scheduling; railway; CPLEX; OPL.

I. INTRODUCTION

The science of better decisions Operations Research (OR) is the discipline of applying advanced analytical methods to make better decisions [2]. By using techniques such as mathematical modeling to analyze complex situations, operations research gives executives the power to make more effective decisions and build more productive systems.

To understand how optimization touches many aspects of everyday life, consider a business trip. The price you pay for your ticket is determined by optimization. The flight crew is scheduled by optimization. Trucks you might see on the road are loaded up and routed by optimization. And when you get to your hotel room, and turn on the TV to relax, the scheduled ads are optimized to maximize the revenues for the network.

The IBM ILOG optimization products put the power of optimization in the hands of business decision makers. We will use these tools for modeling scheduling problem in transportation enterprise.

This work is a logical continuation of [3] and being performed within the scope of the research on the topic "Research and development of innovative unifying models of intelligent systems for the situational response and safety control on the Russian railways", state contract 07.514.11.4039 on September 26, 2011 at lot № 2011-1.4-514-045 "Development of algorithms and software systems for solving problems of exceedingly large scientific data sets storage and processing and data streams collection in real-time" as part of the federal target program activity "Research and development in Russian scientific-technological system 2007-2013 evolution priority directions".

II. OVERVIEW

We classify and give in detail the basic concepts [4]. All vehicles can be divided as follows:

- Aircraft (passenger, cargo);
- Train (passenger, freight, suburban, long-distance);
- Truck (Short-haul, long-haul, hazmat transportation);
- Ship (liner, costal traffic, ferry, tanker);
- Public transit (Bus, metro, suburban train);
- …

Thus, we can identify problems for each type. For example, passenger and public transport planning is contained in the scheduling of the day, while sending orders for truck or rail will be actually in planning for a month. Therefore, according to the duration, the planning can be divided into four types:

- Strategic (several years):
  - Network design and location of main facilities: airports, highways, subway lines;
  - The purchase of the fleet;
- Tactical (several weeks or months):
  - selection of a route bus lines;
  - location of the intermediate objects: stations, warehouses;
  - laying seasonal routes, transportation schedules;
  - prices;
- Operational (a few hours or days):
  - destination transport on the route;
  - transport of parcels;

- Real time (seconds or minutes):
    - Location and movement of ambulances or fire truck

Among these types we are interested only in tactical planning, as time is limited to one month. In the paper we discuss the problem of rail transportation. It can be divided into two classes - passenger and freight. Suburban and intercity trains are all the passenger type. Scheduling is characterized by a single goal for this class. In the case of force majeure on the railway, you can plan bypasses, but it is not as frequent. The special features of passenger transport are:

- Network design;
- Train scheduling;
- Blocking the way;
- Appointment of drivers;

The movement of freight trains is an important part of a fully functioning transportation system. Efficient movement of goods both within and across regions is necessary for industry, retail and international trade, and agriculture. There are airports, major terminals, shipyards in large cities, which are particularly affected by the issue of cargo transportation. The task of planning depends on a very large number of constraints. The peculiarities of this kind of traffic can be classified as follows:

- Fragmentation;
- Not so well organized and optimized, for example, in contrast to the aviation industry;
- The profit can be increased through better planning, use of the crew and pricing policies;

III. TECHNICAL DETAILS

IBM ILOG CPLEX Optimization Studio is one of the IBM ILOG optimization products [5]. These products include IBM ILOG Optimization Decision Manager (ODM) Enterprise and some packaged applications. We don't consider other optimization software, because we need an enterprise application, which simply build with ODM. CPLEX Studio and ODM Enterprise are used to develop custom applications based on Mathematical Programming (MP) or Constraint Programming (CP).

- the CPLEX engine for mathematical programming is used by default when you run your project if your model does not start with the statement using CP;
- the CP Optimizer engine for constraint programming is called if your model starts with the statement using CP;

IBM ILOG CPLEX CP Optimizer is specially adapted to solving detailed scheduling problems over fine grained time [6]. There are, for example, keywords particularly designed to represent typical scheduling model elements, such as tasks and temporal constraints.

With CP Optimizer, you can address the issues inherent in detailed scheduling problems from manufacturing, construction, driver scheduling, and more.

In a detailed scheduling problem, the most basic activity is assigning start and end times to intervals. Scheduling problems also require the management of minimal or maximal capacity constraints for resources over time, and of alternative modes to perform a task.

A. *Interval*

An interval variable represents an interval of time during which something happens (for example, a task occurs, an activity is carried out) and whose position in time is an unknown of the scheduling problem. An interval is characterized by a start value, an end value, a size and intensity. The length of an interval is its end time minus its start time.

An important additional feature of interval variables is the fact that they can be optional; that is, one can decide not to consider them in the solution schedule.

```
dvar interval <taskName>
        [optional[(IsOptional)]]
        [in StartMin..EndMax]
        [size SZ | in SZMin .. SZMax]
        [intensity F];
```

For example, the following variable:

```
dvar interval garden optional in 20..32 size 5;
```

means, that the task "garden", if it will run takes 5 time units and must begin after 20 units of time and before the end 32 units of time.

B. *Cumulative function*

In scheduling problems involving cumulative resources (also known as renewable resources), the cumulated usage of the resource by the activities is usually represented by a function of time. An activity usually increases the cumulated resource usage function at its start time and decreases it when it releases the resource at its end time (pulse function). For resources that can be produced and consumed by activities (for instance the content of an inventory or a tank), the resource level can also be described as a function of time; production activities will increase the resource level whereas consuming activities will decrease it. In these types of problems, the cumulated contribution of activities on the resource can be represented by a function of time and constraints can be modeled on this function, for instance a maximal or a safety level.

```
cumulFunction <functionName> =
        <elementary_function_expression>;
```

where <elementary_function_expression> is a cumulative function expression. This expression includes:

- step;
- pulse;
- stepAtStart;

- stepAtEnd.

Let us consider each of them.

1) *Pulse*

Pulse represents the contribution to the cumulative function of an individual interval variable or fixed interval of time. Pulse covers the usage of a cumulative or renewable resource when an activity increases the resource usage function at its start and decreases usage when it releases the resource at its end time.

```
cumulFunction f = pulse(u, v, h);
cumulFunction f = pulse(a, h);
cumulFunction f = pulse(a, hmin, hmax);
```

The pulse function interval is represented by a, or by the start point u and end point v. The height of the function is represented by h, or bounded by hmin and hmax.

To illustrate, consider cumulative function of using resources, which a volume is a measure. There are two intervals – A and B – which are time-limited. Each interval increases the value during duration.

```
cumulFunction f = pulse(A, 1);
cumulFunction ff = pulse(B, 1);
```

The functions are shown at Figure 1.

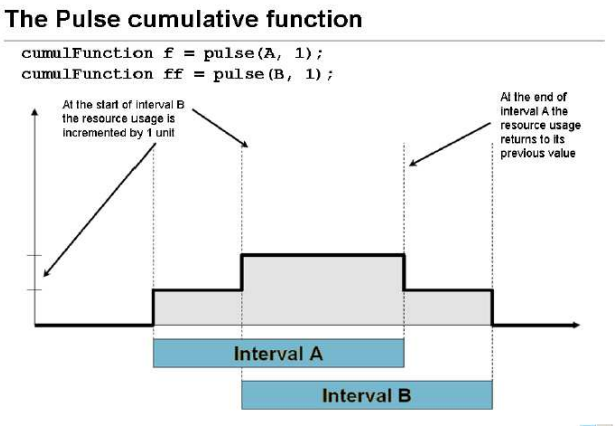


Fig 1. Pulse function

2) *Step*

Step is an elementary cumulative function expression representing the contribution starting at a point in time. Step covers the production or consumption of a cumulative resource.

```
cumulFunction f = step(u, h);
```

where time u is the start of production or consumption and the height of the function is represented by h.

As another example, consider a function of flow measurement resources, similar to the budget:

The level of resources is equal to zero up to time 2, when the value increases to 4.

```
cumulFunction f = step(2, 4);
```

There are two intervals – A and B, which are fixed in time. Interval A reduced level of resources for 3 at the beginning of the interval.

```
cumulFunction ff = stepAtStart(A, -3);
```

Interval B increases resource level for 2 at the end of the interval.

```
cumulFunction fff = stepAtEnd(B, 2);
```

The functions are shown at Figure 2.

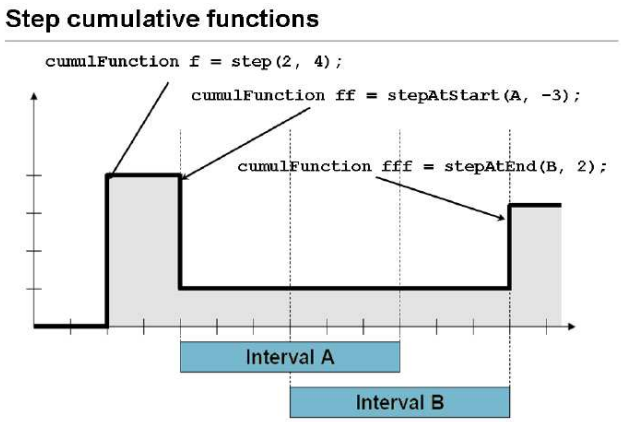


Fig 2. Step function

3) *Alternative*

The constraint alternative(a, {b1, .., bn}) models an exclusive alternative between {b1, .., bn}. If interval is present then exactly one of intervals {b1, .., bn} is present and a starts and ends together with this chosen one.

The constraint alternative(a, {b1, .., bn},c) models a selection of c intervals in the set {b1, .., bn}. If interval is present then exactly c intervals in {b1, .., bn} are present and a starts and ends together with these selected ones. If it is absent, then all b intervals are absent. This constraint is typically used to model the selection of 1 resource (or c resources) among a set of candidate ones. It can also be used in more complex cases to model alternative execution modes for activities or alternative time-windows for executing a task.

The array B must be a one-dimensional array; for greater complexity, use the keyword all.

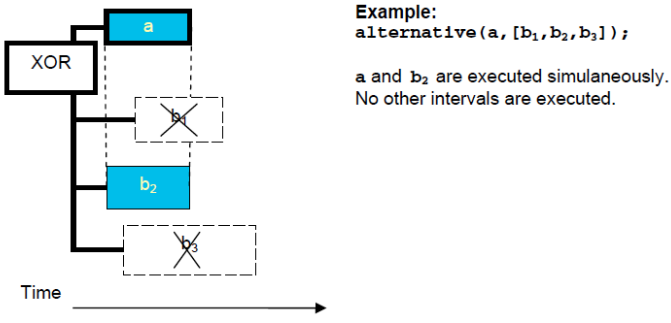


Fig 3. Alternatives

IV. EXPERIMENTAL EVALUATION

The company has available near 100.000 wagons, which are evenly distributed over at the 1.000 stations. There are 10.000 applications for transportation of cargo from one station to another in the company. We should construct transportation plan for a month.

This condition is a real problem. For demonstration we reduce the scale of 100 times. Also part of constraints simplify to understanding of the problem: we aren't considered options such as progressive rate, rate group and so on.

For this problem, we build three optimization models and analyze them. Each of the following models will be an extension of the previous model. For all cases, we will have the same data.

In our case, we have the following transportation network shown in Figure 4. The edges represent the distance (in days) between the two stations, and the figure at the vertex – the number of wagons the first day of planning.

We will answer a few questions about building a model:

- What is the purpose? – Maximization profit for cargo transportation.
- How is the solution? – The decision depends on the time of commencement of the application and the number of wagons.
- What are the limitations imposed on the model? – Number of wagons:
  - Do not exceed the total amount.
  - Should not be below a certain predetermined threshold at the station.
  - Should not be above a certain threshold at the station.

A. *«With returns»*

The first model is called «Order fulfillment with returning». Each order consists of two parts:

- Wagons are sent to the destination station.
- After that wagons will be sent to the original station.

Thus, each order is like a pendulum, has double length of time.

1) *Decision variables*

Decision variables of this model could be identified as two arrays of interval variables. First – Applications:

```
dvar interval app[a in OrderID] optional;
```

We point out the applications are not required. This is due to the fact that not all of them can be performed (total number of wagons in orders at one station may not exceed the number of wagons on this station). The following is a list of identification numbers of applications to find a solution among the set of alternative. As we pointed out above, we consider doubling time of the order:

```
dvar interval alt [o in allOrder] optional in
  o.order.StartData..(o.order.FinshData +
o.place.Duration) size o.place.Duration * 2;
```

We include cumulative function to decision variables. It will take into account the number of wagons at the station in time:

```
cumulFunction count [l in Locations] = step(0,
l.Units) - sum (ao in allOrder : l.Station ==
  ao.order.From) pulse (alt[ao], ao.count);
```

2) *Objective function*

Since the goal is to maximize profits for its implementation, we will summarize the number of wagons multiplied by travel time:

```
maximize sum (ao in allOrder) presenceOf(alt[ao])
      * ao.count * ao.place.Duration;
```

3) *Constraints*

This model is a classic example of vertical alternative:

```
forall (d in OrderID) alternative (app[d], all (ao
   in allOrder : ao.order.ID == d) alt[ao]);
```

Also, number of used wagons at any time shouldn't be less than zero:

```
forall (l in Locations) count[l] >= 0;
```

4) *Result*

This method finds the optimal solution if and only if all kinds of data are distributed evenly. If the number of applications for transport to a particular station is large enough, and the number of wagons on it is a little as we pointed out above, the decision will be far from optimal. But in the original sample data, the optimal solution was found quickly. We get a ready-formed schedule by day, which we could see at figure 5.

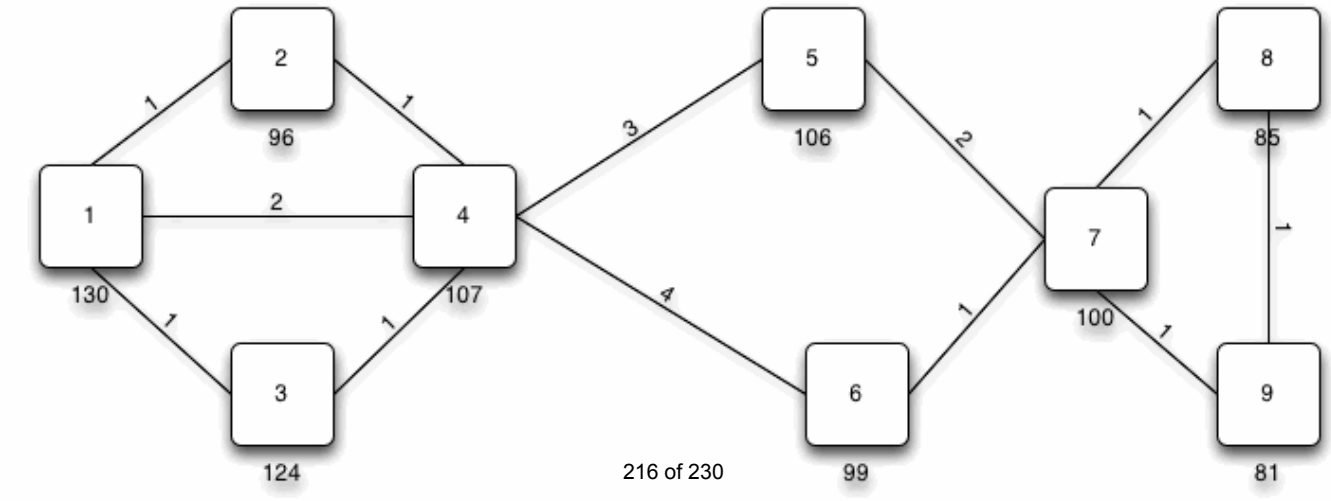


Fig 4. Example net

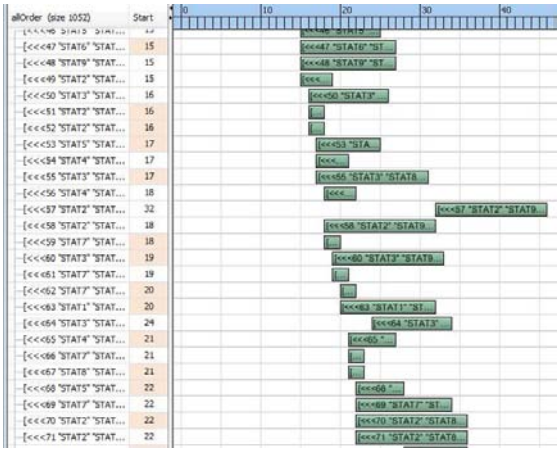


Fig 5. Scheduling

Although we found the optimal solution, this solution is not acceptable. We have to spend a certain amount on the empty wagons run to the station. Thus, the profit will be:

```
Profit = FullPrice – EmptyRun
```

This model, due to the rather large losses in empty run is not applicable in the real world; however, it is the launching pad for the next model.

B. «Without returns»

If in the previous model constructed after the wagons went back, it is logical to assume that these wagons can be left at the station of arrival. Thus, we will not spend their finances on the empty run.

1) Decision variables

Decision variables are changed in relation to the amended model. All alternatives will not have twice the range and will be similar to the input data:

```
dvar interval alt [o in allOrder] optional in
o.order.StartData..o.order.FinishData size
            o.place.Duration;
```

Also, the changes will affect the cumulative function of counting wagons at the station, as the wagons do not come back:

```
cumulFunction count [l in Locations] = step(0,
                    l.Units)
  - sum (ao1 in allOrder : ao1.order.From ==
l.Station) stepAtStart(alt[ao1], ao1.count)
   + sum (ao2 in allOrder : ao2.order.To ==
 l.Station) stepAtEnd(alt[ao2], ao2.count);
```

We are using stepAtStart function considering that the part of wagons from current station run at the beginning of the application. It is a similar situation with stepAtEnd, when wagons come to station after fulfills the order.

The objective function and constraints in this case will not change as we change only the behavior of wagons during the execution of orders.

2) Results

Similarly to the previous model, the optimal solution will be found only when data have uniform distribution. For example, the solution was found, shown in Figure 6:

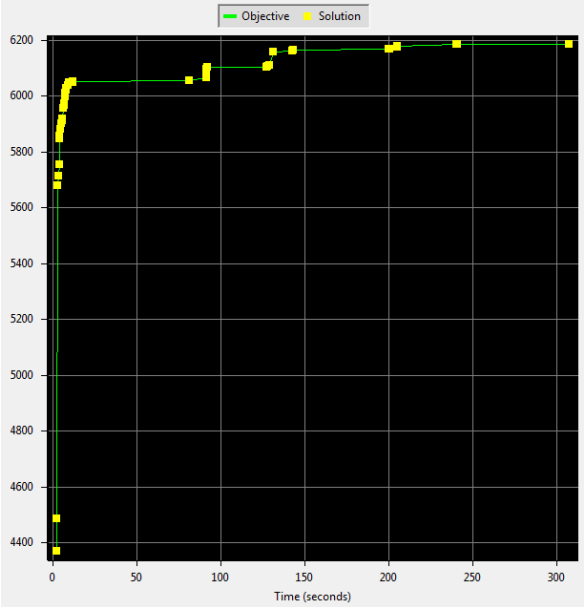


Fig 6. Optimization process

This solution is not optimal, but the ratio of the solution found to the absolute solution will be acceptable for cargo companies:

```
6185 / 6310 = 0,98019…
```

We will get 98% of the maximum possible profit. It can be considered an excellent option.

The drawback of the model is inability to drive wagons from nearby stations. If one station has a lot of applications, then the wagons don't come to the station. As a result, some orders will not be performed. Figure 7 shows the number of wagons at each station everyday. You may notice that the station 2, 5 and 7 have an application for a large number of departing wagons, so for these stations, the number of wagons is almost zero at the end of the month. In contrast, for example, for stations 1, 6 and 8, the number of wagons is increased by 1.5 or even 2 times.

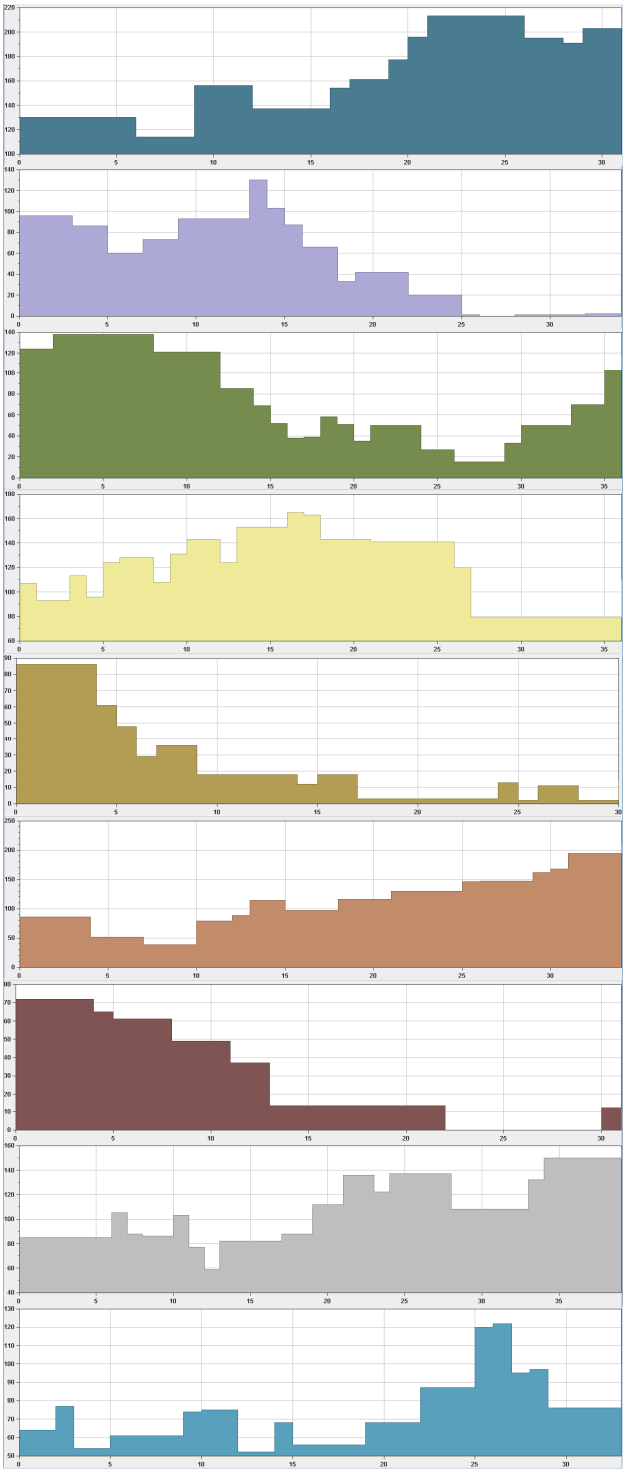


Fig 7. Number of wagons at stations after optimization

C. «Additional empty run»

If some station doesn't have enough wagons for the application, it is reasonable to drive the number of missing wagons from nearby stations. Maybe this will help even increase the objective function.

1) Decision variables

In this case, decision variables augmented with one more variable interval. It has the following type:

```
{EmptyRun} emptyRun = {<do, s, i> | do in
detailOrder, s in Places, i in 0..do.order.Max : s.To
== do.order.From && s.Time <= days};
```

Variables simulate all possible combinations of sending empty wagons from the neighboring stations. So we put the restriction that wagons can be sent from the stations, in which the stage is not greater than a special parameter.

```
dvar interval emptyRuns [o in emptyRun] optional
in (o.order.StartDate - o.place.Duration >= 0 ?
  o.order.StartData - o.place.Duration : 0)..
  (o.order.FinishDate - o.place.Duration) size
            o.place.Duration;
```

Cumulative function is changed to:

```
cumulFunction count [l in Locations] = step(0,
                  l.Units)
   - sum (ao1 in allOrder : ao1.order.From ==
l.Station) stepAtStart(alt[ao1], ao1.count)
    + sum (ao2 in allOrder : ao2.order.To ==
 l.Station) stepAtEnd(alt[ao2], ao2.count)
   - sum (er1 in emptyRun : er1.place.From ==
l.Station) stepAtStart(emptyRuns[er1], er1.count)
    + sum (er2 in emptyRun : er2.place.To ==
 l.Station) stepAtEnd(emptyRuns[er2], er2.count);
```

We add a similar sum to account for incoming and outgoing wagons on the empty runs.

2) Objective function

All empty runs company pays for itself, so the empty run is subtracted from the profit:

```
maximize sum (ao in allOrder) presenceOf(alt[ao])
  * ao.count * ao.place.Duration * FullPrice
                  - sum (er in emptyRun)
    presenceOf(emptyRuns[er]) * er.count *
       er.place.Duration * EmptyPrice;
```

Add the following parameters: price for transportation cargo and price for empty run. Thus the objective function is an order of magnitude higher than the result of the previous examples.

3) Constraints

There are reserve wagons at the station, we make a new restriction. We establish that the number of cars can be in the range [20; 160] at the station at any time:

```
forall (l in Locations) {
alwaysIn(count[l], 0, 60, 20, 160);
}
```

4) Results

This solution is closer to the optimal value, even with restrictions on the number of wagons:

```
311380 / 315500 = 0,98694…
```

This model does not depend on the location of wagons and we can just say that all orders will be fulfilled. It is as close to real use.

V. CONCLUSIONS

This article demonstrates the easy modeling of though simplified, but at the same time, the real problem. We considered several models and provides a comparative analysis. This is just the first step of constructing solutions of the system response and planning. This IBM's tool allows one to quickly build a model of a problem and get an optimal answer. In the future, these developments will be used for detailing that problem. In particular, several large companies are already interested in this. Of course IBM ILOG Cplex Optimization Studio is not designed for building large applications. The next step is to build an application based on the product IBM ILOG Optimization Decision Manager Enterprise, which is precisely designed for that purpose.

VI. REFERENCES

[1] Yudin, DB and Holstein, EG (1961). Objectives and methods of linear programming. Publ. "Soviet Radio".

[2] The transportation planning process: key issues. A briefing book for transportation decisionmakers, officials and staff.

[3] Avdoshin S., Gorbatovskiy M., Chernov A., "The Concept of intellectual situational response system and railways safety of modern Russia", Business Informatics №4 (18), 2011.

[4] Laporte, G. "An overview of transportation planning problem. HEC Montreal, Chaire de Recherché du Canada en Distributique".

[5] Model development with IBM ILOG CPLEX Optimization Studio V12.2.

[6] Detailed scheduling in IBM ILOG OPL with IBM ILOG CP Optimizer.

EnergoWatcher – the platform for creating adaptable energy monitoring systems

Evgeniy A. Kalashnikov
Department of Software and Computing Systems Mathematical Support
Perm State University
Perm, Russian Federation
E-mail: keatrance@gmail.com

***Abstract*. Energy monitoring systems are widely spread among large productions and smart houses. It's designed to collect, process, store data and provide users with aggregated and detailed information about the consumed resources. However, the developing price of these systems is large enough. It is required to have a technology for fast and cheap creation of power consumption monitoring system. These systems should be adaptable to varying of operating conditions and designed for small objects. EnergoWatcher is a development technology of program complexes of power consumption monitoring, which is described in this article. The systems created by means of this technology suppose dynamic adapting to varying of operating conditions and users requirements. The software provides possibility of connection to various sources of the information (to controllers of technical state of objects, to databases of the external applications, and so on).**

Domain-specific modelling; monitoring; metamodelling; power consumption; EnergoWatcher;

I. INTRODUCTION, PROBLEMATIC

This paper is devoted to the power consumption: a lack of information is the main cause of wasteful use of resources, since you can only effectively manage those that can be measured.

The problem of the resource consumption dynamic opacity can be achieved through creation and using **monitoring systems** [1, 2] that allow you to automate the process of collecting data from external sources (sensors, meters, indicators and so on) to obtain a more complete picture of what is happening. It should be noted that data from the energy accounting devices are handled by software developed by equipment manufacturers to work with specific devices. This specialized software often makes it impossible to integrate data from different sources, processing and presentation in one program. The key objectives of monitoring systems are to collect, process, store data and provide users with aggregated and detailed information about the consumed resources collected from heterogeneous sources.

It should be noted that the need for such monitoring systems are increasingly seen not only in industrial scale (large production), but at the household level. There are government agencies and private companies seeking to control the level of water / heat / electricity consumption etc. in order to save his own funds. However, the monitoring system - an expensive, "custom made" product, developed under the specific type and configuration of equipment, so there is a problem on the establishment of flexible software, which could be configurable while condition changes of maintenance (connection of new devices, monitoring of certain parameters, etc.).

So it’s necessary to have a technology for easy dashboard construction. Using generative programming approach we could generate monitoring systems with minimum costs without involving programmers. Such software can be reconfigurable for new connected devices and their parameters.

The main requirement for such technology is the ability to adapt the system constructed under specific operating conditions [3]. The system should be dynamically adjusted to the new hardware and user needs, without the programmer (without changing the source code and subsequent recompilation). To this end, a toolkit is being developed, that allows construct virtual panel for energy consumption monitoring, which in future will be discussed by experts and will lead to relevant policies to improve energy efficiency. In this paper we propose a technique EnergoWatcher - the platform for creating adaptable energy monitoring systems.

II. MONITORING SYSTEMS: ARCHITECTURES, APPROACHES

There are two basic approaches to the development of monitoring systems [4] - "manual" and "automatic". In the first case, the system is programmed from scratch for a specific hardware system architecture; system's components are strongly coupled; the code is optimal in terms of performance. However, it would lead to significant development costs, inflexible solutions and inability to reconfigure the system when adding new hardware.

While using an automatic generation of software development the code and database generation happens on the basis of the system’s model defined by developer. This is the standard approach with CASE-tools [4], based on transformations of the original model of monitoring system in code. In this case, we have a universal solution to the high rate of development to the detriment of generated code performance. However, this solution is not flexible too,

because the slightest change in the architecture of complex hardware / system requirements will need to re-generation of source code and database.

In both approaches adapt facilities of the final applications are limited. The specificity of the modern software is that the application life cycle never ends with the completion of its development. Therefore it is required to provide the ability to dynamically configure system to new operating conditions, the needs of its users.

To create an adaptable system of this kind is needed to store a description of the hardware configuration (new connected devices, parameters, their tracking), and other settings. If you change the configuration, the application code should not be changed. The program functioning is based on the interpretation of these descriptions (metadata) [1]. User interface (UI) is generated based on this metadata every time when you run an application, what makes possible to construct and keep UI settings. That mode of interpreting the metadata allows achieving the necessary flexibility and balance between the rate of development, flexibility and adaptability of the application possibilities.

III. ARCHITECTURE OF ENERGOWATCHER SYSTEM

The core of the system is metadata that describes components of the system from different viewpoints [2]. Metadata's model (meta2model) of logic level is the main. It is based on the ER-model of Chen (entity-relationship model). Each entity is an abstract real-world object (e.g., meters), with its own properties (attributes) and relationships with other entities (links). Instances of the entity represent the existing concrete objects and their attribute values and relationships reflect their current state.

Let's consider the EnergoWatcher's application architecture, the multi-level monitoring system. The system consists of five logical and physical levels (see Fig. 1).

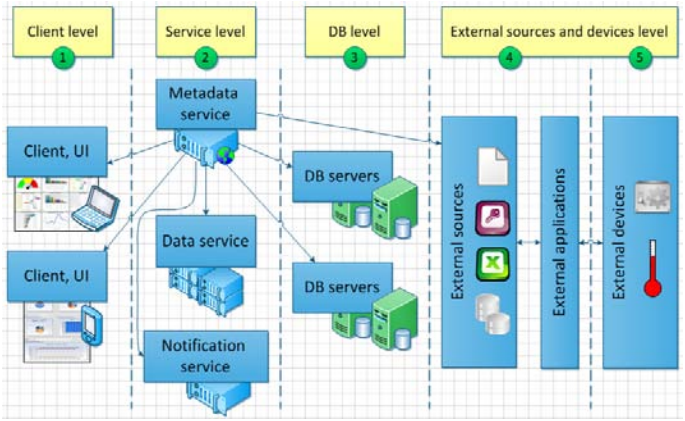


Figure 1. Levels in EnergoWatcher's architecture

Levels 4 and 5 are responsible for connecting to external sources, configuring the import data parameters. The data, which obtained with the metering of energy consumption, are handled by foreign applications. After processing the received device parameters are stored in external sources (databases, spreadsheets, CSV files, etc.). The system asks for new sources of data at defined intervals of time. If this data is available, the system processes import into the database (level 3) caching obtained values at the same time [5].

For each configuration of hardware on database server (level 3) based on the metadata its own database is created to store rates selected for import. The database structure is determined by the metadata of logical and physical level (Fig. 2) in accordance with the existing hardware configuration. It is updated automatically at metadata changing via component restructuring.

The components of levels number 1 and 2 are part of a client-server service-oriented architecture. The end-client establishes a connection with the services in polling or duplex mode and receives data (data services, and events) and metadata (metadata services and settings UI). Metadata service is available on user request and provides metadata of logical level. UI service, which is actually part of metadata service, represents metadata of presentation level. Event service notifies connected clients automatically if emergency situations arise. Data service sends new data when new imported values appear on server side. The client defines its own interface of dashboard and configures the connection UI controls to data sources (binding) in return.

The logical connection between these levels is provided by interdependent layers of metadata (MD), describing the system from different angles (Fig. 2). Key metamodel - a model of logic level - provides a link of metadata and system operation under terms of a specific domain for the user (in this case in the field of energy auditing).

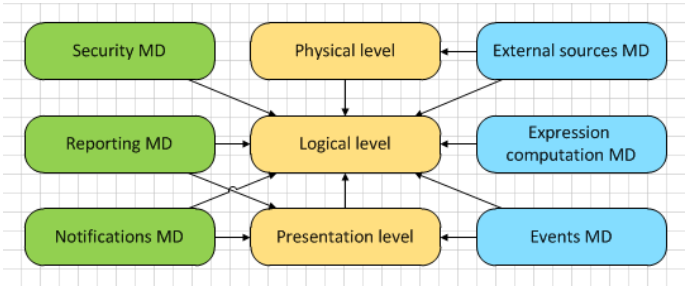


Figure 2. Interdependent metadata (MD) layers

IV. REQUIREMENTS FOR UI SYSTEM

One of the main goals of the monitoring applications is a convenient and timely notification of the ongoing processes, emergency situations, displaying the real picture of energy consumption (energy audit) for further analysis [6]. It's all about the data visualization, which perhaps has the most important value in operational dashboards.

In general, to create own interface user (or administrator) will need to perform several actions:

- create instances of the visualization components (controls);
- place these controls in the required locations, given their mutual nesting and hierarchy;
- configure the basic properties of the controls;

- specify the data sources for display - those attributes, whose values you want to display, and, if necessary, set the missing values.

To ensure maximum system flexibility and adaptability to dynamically changing the operating conditions were determined following requirements for UI system [7]:

1. Setting up the visual interface should be simple and understandable to the end user-nonprogrammer:

- using of intuitive action in WYSIWYG way: Technology Drag & Drop, the component is resized with the mouse, etc.;

- a set of display properties for component configuring and their descriptions should be dynamically configurable - in fact, they become part of the metadata system. It is required to display user only necessary properties and group them into predefined categories;

- the problem of localization of all the properties of the components need to be solved: they need to be shown in the user native language (in our case, in Russian) when configuring the component. Thus, ordinary users are unlikely to know what the terms mean "Width" and "Height";

- meta-description of the parent control properties (within the concept of the OOP) to be inherited by child controls with the ability to override them (to avoid massive duplication of descriptions and provide illustrative descriptions).

2. The set of supported components to be dynamically extensible:

- adding a new component must be clear to the user without data addition in the database;

- format for storing information on the control and its properties should be open – e.g., XML;

- .NET component of visualization must have clear interfaces to implement the logic to display the incoming data and, if necessary, entry of missing values on data service.

3. The UI components must support one of the possible types of data binding:

- none binding: component is used only to display static information and / or grouping other components;

- single binding: component displays only one parameter of the system over time (e.g., reading a thermometer);

- multi binding: component displays multiple parameters of the system over time (e.g., a graph with multiple indicators).

4. All changes in the system should be applied at run time, without recompiling the source code, using the principle of WYSIWYG. In many systems, the interface is not the model is interpreted at runtime, but only serves to generate user interface code [6, 8].

It should be noted that the controls themselves, as well as connected data sources, should be able to be dynamically connected to the system without recompiling the source code.

V. PRESENTATION METADATA AND BINDING COMPONENTS

In EnergoWatcher technology, based on the interpretation of interrelated metadata, user creates his own dashboard, described by the metadata of presentation layer (Fig. 3). User selects indicators what he wants to watch and sets their configuration, using the means of creating a visual interface of customizable panel in EnergoWatcher.

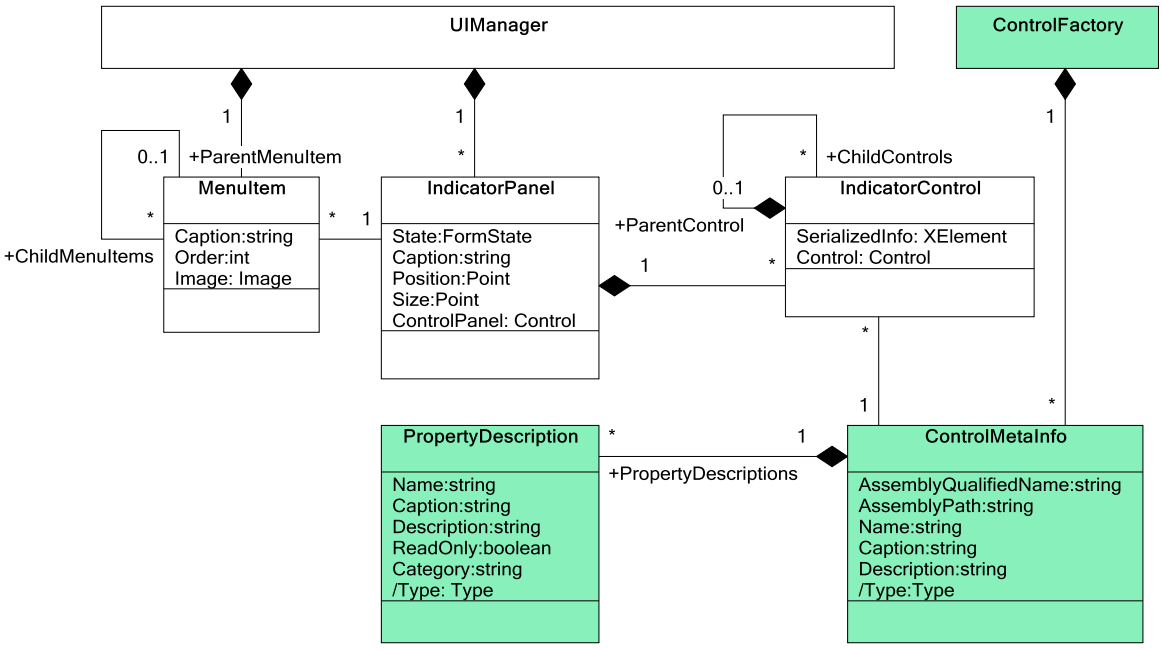


Figure 3. Two parts of presentation layer metadata

For the dynamic connection management components and configuration UI presentation layer metadata is divided into two levels: UI level and level of the interface components.

The UI level (see Fig. 3, classes without background) describes the user's desktop: tree menu (menu items *MenuItems*), and a lot of dashboards *IndicatorPanel*. Dashboard is a collection of custom visual components *IndicatorControl*, located in the hierarchy relative to each other. Each component has its own internal serialized state, which is used to saving and restoring the configuration user interface. During system operation the visual components are dynamically linked to the sources, the values in which come from the server through a duplex channel.

Component level (see Fig. 3, green background) contains meta-information about the components of imaging: an indication of the type and assembly, in which the control is located, as well as a list of descriptions of the component properties. Meta-descriptions are stored in XML file, resulting in the connection of new components takes place directly at run time, besides the issues of descriptions localization and names of configurable properties are resolved.

As mentioned above, the presentation model has a connection with logical model. It provides binding UI components to server sources. There are 2 kinds of objects in the capacity of server sources: attributes and expressions (see Fig. 4). Every time when the application starts, controls establish communication with *LightSource* element. Server metadata [1] of available sources is too heavy to be transported to client, that's why there is light representation of them (it is realization of pattern "Data Transfer Objects", DTO).

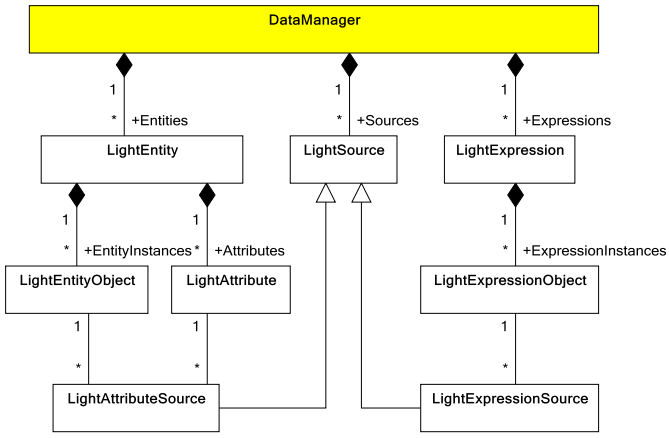


Figure 4. Client metadata DTOs for data management

LightAttributeSource represent values of attribute *LightAttribute* of instance *LightEntityObject* of entity *LightEntity*. *LightExpressionSource* represent values of expression instance *LightExpressionObject* of expression *LightExpression*, which is computed on server. Actual data is collected from server automatically using duplex mode of service communication.

CONCLUSION

This paper proposes an approach to building monitoring systems, based on an interpretation of metadata in real time. The article describes presentation model of EnergoWather system for building user interfaces in runtime by WYSIWYG way.

At this moment system are being developed and it is planned to product a prototype version of EnergoWatcher. It's need to implement set of subsystems:

- operations with event metamodel, including describing specific events in domain specific language (DSL);

- straight connections to devices in real time using standart transport protocols;

- data analizer for automatic recognition of leaks and other emergency situations;

- services for supplying data, events and metadata to clients;

The system was tested at the tenders U.M.N.I.K. and B.I.T., nowadays it is in developing state. It's planned to integrate it to municipal authorities and resource-demanding facilities and provide the power consumption audit.

The proposed system focused on monitoring in order to save own funds. First of all, it will be interesting to introduce EnergoWather in small industrial and social facilities (schools, hospitals, kindergartens), as well as other municipal property objects. Generated software will analyze the dynamics of daily energy consumption and compare it with historical state. So it is possible to find an illegal consumption size as well as to track the occurrence of abnormal situations.

REFERENCES

[1] V.A. Voronov, E. A. Kalashnikov, L.N. Lyadova Technology of development of power consumption monitoring system // Proceedings of the Congress on Intelligence Systems and Technologies "AIS-IT'10". Scientific publications in 4 volumes. Moscow: Physmathlit, 2010, Vol. 4.

[2] E. A. Kalashnikov Tools for creating dynamic dashboards of energy consumption on the basis of related metadata models // III All-Russian Student Research Forum (electronic conference, http://rae.ru/forum2011/104/285).

[3] Joseph W. Yoder, Ralph E. Johnson: The Adaptive Object-Model Architectural Style. WICSA, 2002, pp. 3-27.

[4] K. Czarnecki, U. Eisenecker Generative Programming: Methods, Tools, and Applications. Reading, MA, USA, Addison-Wesley, 2000.

[5] E. A. Kalashnikov Caching in the systems for monitoring of energy consumption // IV All-Russian Student Research Forum (electronic conference, http://rae.ru/forum2012/219/2697).

[6] Ekkerson U. Performance Dashboards: Measuring, Monitoring, and Managing Your Business. Moscow: Alpina Business Books, 2007.

[7] E. A. Kalashnikov Creation user interface in WYSIWYG way for energy consumption indicators monitoring system. Math of software system, Perm University, Perm, 2010, pp. 84-91.

[8] Mohan R., Kulkarni V. Model Driven Development of Graphical User Interfaces for Enterprise Business Applications // MODELS'09 Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. Springer-Verlag Berlin, Heidelberg, 2009, pp. 307-321.

Development experience of ore extraction and traffic simulation system in potash mines – bundled software “Рудопоток”

Gleb Chudinov
Perm State National Research University
Perm, Russian Federation
e-mail: gleb.chud@gmail.com

E.B. Zamyatina (research supervisor)
Perm State National Research University
Perm, Russian Federation

Abstract— **Development experience of ore traffic and extraction simulation system in underground part of potash mines is reviewed. Distinctive features of suggested simulation model (combine complex extraction, using conveyor transport and transfer point in continuous ore traffic simulation) are represented. Implementation features of bundled software ”Рудопоток” are considered.**

Keywords: simulation model, simulation system, potash, mine, extraction, conveyor, ore traffic

I. INTRODUCTION

The final goal of potash mining company is increasing extent of mineral production with simultaneous decreasing of its expenses. Considerable contribution in total bottom-line cost is endowed by ore extraction and transportation in mines’ underground parts. In connection with this problem of optimal mining planning become urgent. It includes tasks for guaranteeing minimal-required mineral content (quality), continual conveyor workload and uniform ore arrival to shafts with total decreasing transportation costs. Additional tasks for solving problem are selection optimal sizes and places for bunkers in transfer points, conveyor transport estimation, determination for transport scheme and layer way of ore extraction.

Significant to note, classical task of statistical rough estimation for combine complex production rates was researched and solved in mining literature, example [1]. But for interactively solving the problem (in dynamic) it needs simulation. Though according to author investigations, also [2] – most of complex models are from 1980s and technically turned old because of computing machinery’ and information technologies rapid development. Modern researches [3] are devoted to coal mines and explore different aspects, laying aside questions of ore quality composition, its extraction in mining sites, and don’t take into account conveyor transport specific and ore flow continuous nature.

The development of detailed simulation model and simulation experiment for ore extraction and transportation in underground part of potash mines and its software implementation (bundled software “Рудопоток”) have been progressing since 2008 year [4]. The project is elaborating in Aerology and Thermal Physics Laboratory of Mining Institute UrB of RAS and it includes the investigation results of Computer Science Department[5] (Perm State University).

The paper demonstrates key features (generic aspects) and main specific characteristics for developed program model (via bundled software) of ore extraction (layered, heading-and-stall method by combine complexes) and transportation using conveyors and transfer points in underground part of potash mines. Suggested model have been verified, validated, calibrated and have been approved on transportation network of mine БКПРУ-4. The methodologies for determination of optimal parameters for transport equipment have been developed.

II. SIMULATION MODEL AND ITS FEATURES

Most of known models in freight (ore) flow [1], [2], [3] are simplified or investigated only part of needful aspects. Often, they don’t take ore mineral quality (composition) into account, it’s impossible to manage conveyor speed and power consumption and to optimize these parameters. Ore movement on conveyors isn’t full and is performed without taking into account the cross-sectional profile of the conveyor belt. Mining sites are replaced with aggregates usually.

One of ways to solve the problem is to use agent oriented paradigm in relation to simulation [6]. In this case simulation model of ore extraction in mine is presented as an extended object scheme for conveyor transport. Structure for the transport network – blocks, conveyors, transfer points and relations between them - is specified in program as user input (Fig. 1).

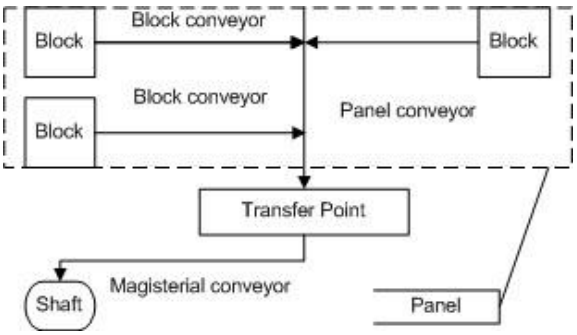


Figure 1. Model of mine transport scheme – common components

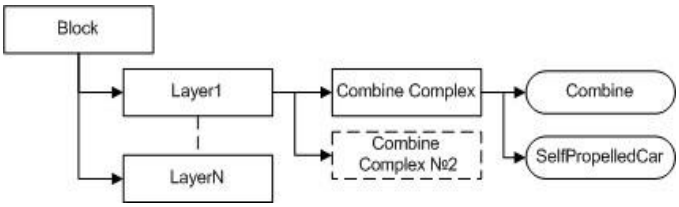


Figure 2. Component structure of mining site (block)

User inputs parameters: belt width, length, speed; side rollers angle of inclination and etc. – for conveyors. Transfer points (TP) are equipped with discharge devices (DD) with specified productivity for each. Mining sites are separated on native parts – multiple layers and combine complexes (Fig. 2). Each unit has its own characteristics. Model takes into account combine complexes' working order (plan), different equipment types (models) and physical restrictions for them; also sizes and frequency of stall and ore passes placement (layout); technical and organizational idle times. All these parameters and some others influence on simulation process and output results.

Important feature is ore transportation on conveyors with layered holding (Fig. 3) and taking cross-section belt profile (Fig. 4) into account. Model implements ore continuous flow transportation process in discrete model (bunker filling and emptying are subject to ratio of DD bandwidths and incoming ore rates). It implements ore proportional unloading on multiple conveyors according to realistic physical behavior and constraints.

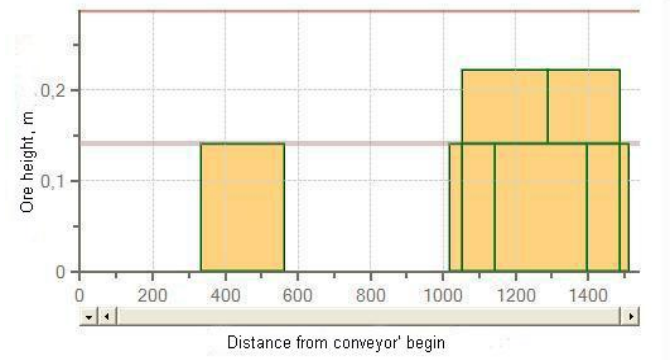


Figure 3. Ore width/height distribution on conveyor belt

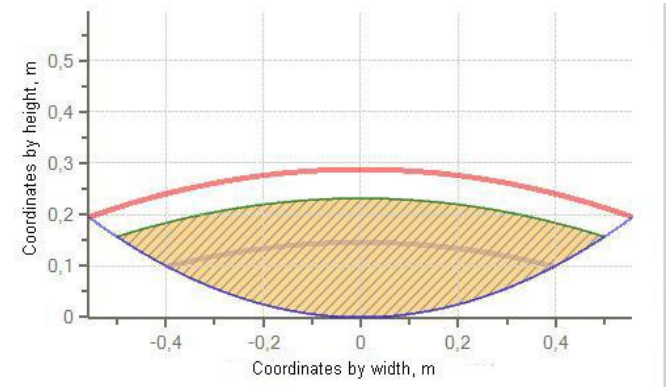


Figure 4. Conveyor belt cross-section profile

Ore mass' mineral composition quality are taken into consideration, it's computed during excavation process in mined layers in block. Model keeps track of mineral content in the cases of ore mixing in bunkers and as result of ore layering on panel (magisterial) conveyors.

Methodology for computation average-required conveyor productivity and minimal-required TP bunker volumes are developed on the base of imitation experiments' series on calibrated model (natural measurements were executed for defining accurate combine complex operations' duration). This methods use mean equipment workload for the time of all simulations running (Fig. 5). Simulation engine takes optimal value for each iteration, and maximum of them from experiments series as final result for equipment.

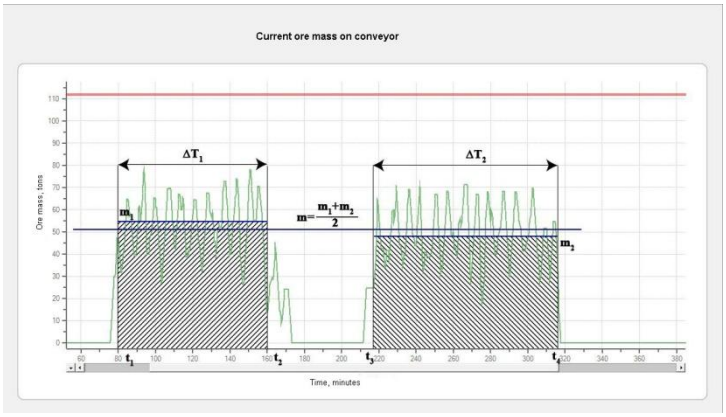


Figure 5. Determination of mean ore mass along conveyor belt

III. HIGH LEVEL ARCHITECTURE VIEW OF "РУДОПОТОК" SOFTWARE

Completed program implementation is interaction along time among three main modules by areas of responsibility: graphical interface (GI), data persistence and simulation model (SM). In turn GI and SM are divided into lesser components.

Graphical interface ensures common user data editing (definition of transport network structure and its main components – blocks, conveyors and transfer points, interconnections between them; parameters of model equipment). Besides, after series of imitational experiments results are displayed into diagrams of ore mass and its mineral composition quality over time, and some others. Animation for processes of ore extraction and transportation over all mine are implemented on basis of received historical data from completed experiments (Fig. 6).

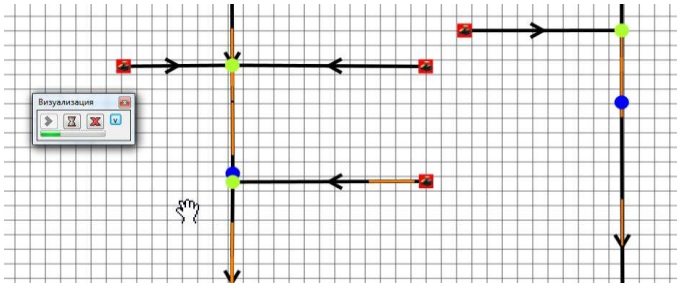


Figure 6. Ore extraction animation on all transport network, blocks disabled

Statistics gathering module is represented by classes for aggregation and storing states of objects over time and gathering statistical information after simulation completion. Basis for optimization is data linearization, when only begin and end of line sections are retained for linear-changing parameters (removing intermediate points as unnecessary) (Fig. 7).

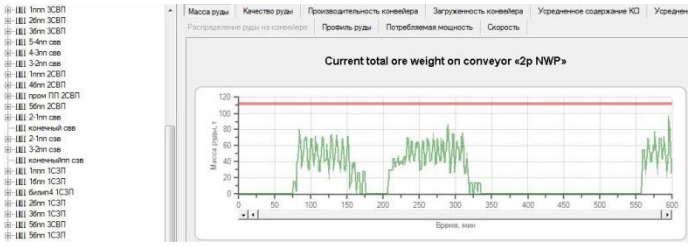


Figure 7. Ore extraction animation on all transport network, blocks disabled

Simulation model consists of domain classes (Fig. 8), time advancement system (scheduler) and statistics gathering module. Domain implements main conception classes such as block, conveyor, transferring point. Equipment behavior along time (combine, self-propelled car) is simulated indirectly on basis of state diagrams (work without outer influence) and activity diagrams (equipment interaction, for example, ore transfer). Time advancement system consists of scheduler, where discrete event algorithm for time advancement with constant and variable step (with possible sub steps) is implemented.

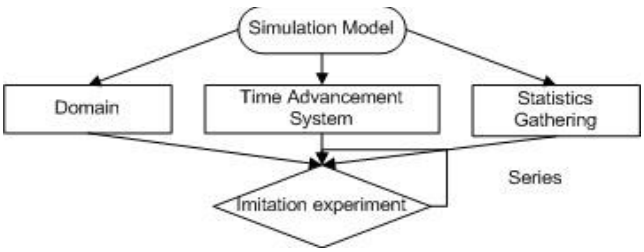


Figure 8. Simulation model main components

IV. Conclusion and Experimental Results

Thus, object imitational model of ore extraction and transportation in underground parts of potash mines is developed. Its verification, validation and calibration have been made. On model basis methods for estimation of equipment optimal parameters (TP bunker volume, conveyor belt bandwidth) are suggested and developed. Model was implemented as bundled computation software “Рудопоток”, which affords means for creating (building), running and analyzing of transport networks (existing or only under construction); animation for processes of ore extraction and transportation in all over the mine in whole. Experimental check has been accomplished on transport scheme for mine БКПРУ-4 on 2010 year.

References.

[1] Лыхин П.А. Добыча калийных солей. (In Russian) URL: http://www.mi-perm.ru/authors/lyhin/b1_text.htm#s

[2] Панасюк И.И. Имитационное моделирование организации рудопотока и управления затратами горнорудного предприятия: дис. канд. экон. наук : 08.00.13 СПб., 2005, 263 с. (In Russian)

[3] Конюх В.Л. Имитационное моделирование системы подземного транспортирования / В. Л. Конюх // Проблемы информатики, 2010. - №3. - С. 43-53. (In Russian)

[4] Круглов Ю.В., Мальцев М.С., Чудинов Г.В. Имитационное моделирвоание горных работ и рудопотоков в калийных рудниках ОАО «Уралалий».// Горный журнал, 2011.- № 11 .- С. 20-22 (in Russian)

[5] Замятина Е.Б., Чудинов Г.В. Разработка и использование программных средств для построения и исследования агентных имитационных моделей. / Е.Б. Замятина, Г.В. Чудинов // Вестник пермского университета. Математика, механика, информатика, 2010, №2(2), С. 80 – 84. (in Russian)

[6] Macal C.M., North M.J. Agent based modeling and simulation / C.M. Macal, M.J. North // In the Proceedings of the 2009 Winter Simulation Conference, ed. Rossetti M. D., Hill R. R., Johansson B., Dunkin A., Ingalls R.G., 2009. P. 86-98.

Research of methods for constructing message-passing interprocess communication based system for railroad situation analysis

Daria Kobyakova
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
dskobyakova@gmail.com

Scientific Advisor: Prof. Sergey Avdoshin
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
savdoshin@hse.ru

***Abstract*—This paper briefly outlines the research of parallel system constructing methods for railroad situation analysis and emergencies prediction. Within the scope of the research various paralleling method are considered and analyzed in terms of application for railroad emergencies prediction.**

***Keywords:* parallelism, emergencies prediction, real-time data mining, exceedingly large volumes of data processing.**

I. INTRODUCTION

Nowadays in railway cargo transportation area contingencies such as crime, theft, locomotive breakdown emergencies are controlled and regulated mostly by dispatcher's offices. And in most cases contingencies become known only after they have happened, what results in the loss of money, time and client confidence. The solution is to collect data both structured and unstructured, originated from a wide variety of sources such as cameras, sensors, news feeds, VoIP and traditional databases, analyze and thereby identify and prevent possible contingency situations or emergencies on the railways. It is clear that the volume of such source data can be enormous and is estimated in terabytes or even petabytes. In order to achieve effective and timely processing of enormous volumes of data real-time and with extremely low latency it is necessary to use parallel algorithms and indeed it is critical to choose an appropriate paralleling method. In this study various parallel system building methods will be analyzed in terms of application in railway situation analysis.

This work is being performed within the scope of the research on the topic "Research and development of innovative unifying models of intelligent systems for the situational response and safety control on the Russian railways", state contract 07.514.11.4039 on September 26, 2011 at lot № 2011-1.4-514-045 "Development of algorithms and software systems for solving problems of exceedingly large scientific data sets storage and processing and data streams collection in real-time" as part of the federal target program activity 1.4 " Research and development in Russian scientific-technological system 2007-2013 evolution priority directions".

II. SPECIFIC TASKS AND OBJECTIVES OF THE RESEARCH

A. Tasks

Main subtasks in my research include the following steps:

- Identify kinds of situations to be predicted

On this stage of the research only one kind of situation is considered: accident due to technical failure

- Generate data sets that are necessary for prediction
- Build a predictive model based on historical facts of railway situations using special data mining software IBM SPSS Modeler. The accuracy of predictive algorithm must be not less than 75%.
- Identify type of parallelism that is typical of the task and choose appropriate paralleling method. It is also supposed that on this stage various paralleling methods will be analyzed in terms of application for railway situation analysis.
- Implement parallel predictive algorithm (parallel algorithm development using special software IBM InfoSphere Streams, testing, debugging).

Preprocessing data preparation steps, including refinement, cleaning, aggregation and transformation are beyond the scope of the paper.

B. Objectives

The result of the research is expected to be a parallel algorithm targeted to analyze situations on the railways particularly:

- Accident due technical failure

III. TYPES OF PARALLELISM

In different kinds of tasks the following types of parallelism usually occur [6]:

- data parallelism

This type of parallelism is typical of tasks that include the repeated execution of the same algorithm with different input data. Such calculations can obviously be done in parallel. If the problem has a parallel data, parallel program should be organized as a set of identical programs, each of which runs on its own processor from the main program. Such a program is usually a coarse grained one. Paralleling method based on data parallelism is called data decomposition.

- functional parallelism

This kind of parallelism is based on different functional blocks in an application. It can be split into separate processing units, that communicate with a fixed number other units in such a way that the output of one part serves as the input of another part. Method of parallelization based on functional parallelism is called functional decomposition.

- geometric parallelism

It requires that the problem space should be divisible into sub-regions, within which local operations are performed. The difference between the geometric parallelism and data parallelism is that in the first one subtasks of processing in each of the subareas must be interconnected. Parallelization based on geometric parallelism is called domain decomposition method.

- algorithmic parallelism

Stands for a type parallelism, which is detected by identifying in the algorithm the fragments, which can be performed in parallel. Algorithmic parallelism rarely generates coarse-grained (large-block) parallel algorithms and programs. The paralleling method based on this type of parallelism is called algorithmic decomposition.

- pipelined parallelism

This type of parallelism is typical of task in which input data must go through several stages of processing. In this case it is natural to use the pipeline decomposition of a task.

- «disorderly» parallelism

Often occurs in classes of algorithms where the possible number of parallel branches and the computational complexity are a priori unknown and depend on a specific task.

IV. PARALLEL PREDICTIVE ALGORITHM DEVELOPMENT

A. The process of model training

All information (data about precedents) required for the model construction is going to be extracted from operational sources to a special file containing of a table with facts hereinafter referred to as full set. The rows in the table represent precedents, and the columns are attributes of each precedent. In the last column there are losses suffered as a result of each situation. Then, in order to build accurate losses- prediction model, two random samples from full set must be selected:

- training (learning) sample;
- control (testing) sample

Building the predictive model on the learning sample will result in a certain function F, mapping X (a set of attribute values for each precedent) to Y - the predicted value of possible losses. Class of the function depends on the learning technique. Some of such techniques are supposed to be considered during the analysis:

- Logistic regression
- Neural Networks
- QUEST
- Decision-tree
- C&R Tree

To assess the quality of the obtained model it must be run on the testing sample, then the predicted losses will be compared to the actual damage by calculating special metric ROC, which shows the accuracy of predicted values or average forecast error [2]. However there is still the risk that the obtained error may be strongly dependent on how the full sample has been split in learning and control ones. Therefore, the next step should be so called cross-validation. Cross validation is a statistical method of evaluating and comparing learning algorithms that assumes that full sample should be divided into subsets for training and validation randomly multiple times [3]. 10-fold cross validation is commonly used [4]. For each split must be calculated ROC. Then averaging will be used in order to estimate prediction error of each learning technique. The algorithm that comes out best (minimal) average ROC is considered as superior to the other one.

B. Parallelization in model training process

In the posed task parallelism occurs not only during real situations prediction but also on the model developing stage.

1. Predictive model development

In real practice the size of the full sample is usually too large

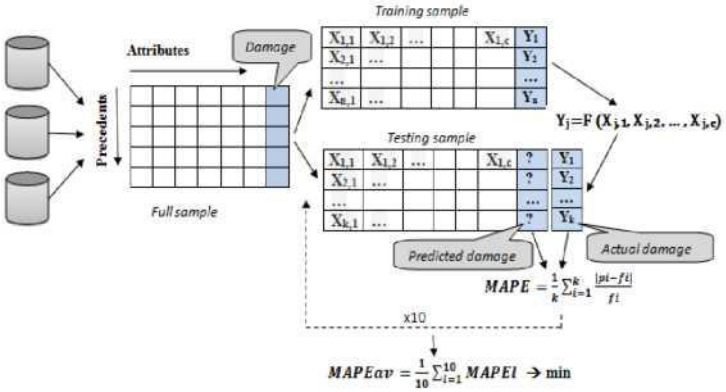


Fig. 1. Predictive model development process

therefore all the calculations can take a long time, what does not meet the target of timely response. The solution is to compute ROCs for each split in parallel. Thus *data parallelism* is typical of the posed task as it includes the repeated execution of the same algorithm with different input data. Algorithm parallelizing should be performed using data decomposition method. The process on the predictive model development will be organized as a set of identical processes, each of which runs on his slave processor from the main program that runs on the master processor.

C. Predictive model selection

The initial set of features is as follows:

(<Part of the railroad>,< the average weight of trains passed through the area over the past day>,< device 1 serviceability>,< device 2 serviceability, temperature>)

We will train the model and select the most important for prediction features simultaneously. Training of the models is supposed to be carried on in 4 steps progressively complicating the relationship among the data.

Step 1. Firstly it's necessary to generate the initial set of five features. Dependencies should not be trivial, so assume that information about the serviceability of the device 2 is not always reliable (the device fails): sometimes the device 2 serviceability indicates “true”, but really it has “false”. In 80% of cases with such a failure railway accidents happen. The rest of the accidents occurs due to unknown reasons. Failure of the device in 50% of cases are due to too low or high temperatures. The remaining data do not have any explicit dependencies. The results of training the model on such data set are presented in table below.

	Decision tree	Neural Networks	Logistic regression	QUEST	C&R Tree
Training accuracy	0.331	0.821	0.792	0.842	0.848
Testing accuracy	0.333	0.815	0.787	0.829	0.837

According to the results decision tree algorithm will be excluded of the further selection process, because it's results on

the very first sample are unsatisfactory.

Step 2. On this step it's necessary to complicate the task of modes learning by increasing the number of predictors. Select most important for the prediction features from those available. In almost any problem of forecasting the question arises: what signs to use, and what not. The problem of features selection often arises from the fact that at the stages of formulation of the problem and the generation of data is not yet clear what the signs are prediction-useless or duplicate each other. The challenge for feature selection in its exhaustive search nature. If the number of sign is n, the number of non-empty subsets of 2n -1. Direct enumeration of all subsets is impossible if n is the order of 20 even in the most modern machines. Attributes synthesis (also called features extraction) is the approach to reduce dimensionality. It consists of finding a transformation of the original feature space into a new space of substantially smaller dimension. One of the well-known and frequently used methods is the sequential addition of features - ADD method. This method assumes adding to an existing set of one additional feature, and the choice of ones, which leads to the greatest predictive error decrease (or predictive model accuracy increase) on a testing sample. It should be noted that the ADD method reduces the complexity of brute force but sometimes it tends to include a set of extra (noise) features. In our case, the duplication of features can be neglected, because we have just the model of the real situation so it's simplicity assumes the minimum number of data set. All features available are:

(<Part of the railroad>, <average mass of trains passing through particular sector per day>, <device 1 serviceability>, <device 2 serviceability>, <temperature>, <device 3 serviceability>, <ware of contact wire>, <DISC - sensor reading>, <maneuverable light signal>, <input light signal>, <occupation intensity per day>).

For best features selection, we take the predictive algorithm, which showed the best results in the first step –C&R tree.

After adding to the existing data set “device 3 serviceability” feature algorithm re-training came up with following results:

	C&R Tree
Training acuracy	0.848
Testing accuracy	0.845

Note that the accuracy of the model on the training set on average has not changed, but the accuracy of prediction on the testing sample increased. We conclude that this feature is not excess and leave it in the set of predictors. The next feature that we will check - the “wear of contact wire”. The result of the experiment is shown in the following table:

	C&R Tree
Testing accuracy	0.875
Training accuracy	0.874

The results indicate that this predictor is surely quite informative so leave it in the sample too.

After the sequential adding of the next 5 features the accuracy was practically unchanged, but after adding station battery indicator feature, the prediction accuracy was reduced, so this feature will not be considered in further process of algorithms training. Final sample get the following attributes:

(<Part of the railroad>, <average mass of trains passing through particular sector per day>, <device 1 serviceability>, <device 2 serviceability>, <temperature>, <device 3 serviceability>, <ware of contact wire>, <DISC - sensor reading>, <maneuverable light signal>, <battery voltage>, <input light signal>, <occupation intensity per day>).

The results of the re-training on the obtained sample are as follows:

	Neural Networks	Logistic regression	QUEST	C&R Tree
Training accuracy	0.862	0.790	0.849	0.875
Testing accuracy	0.857	0.786	0.848	0.875

Logistic regression came up with less accuracy than on the previous step. Since the following training stages are supposed to have more complex relationships this model will not be considered.

Step 3.Then complicate our relationships, and suppose that an accident occurs in only 50% of the device 2 failures, other emergencies do not have explicit dependencies.

	Neural networks	QUEST	C&R Tree
Training accuracy	0.884	0.892	0.928
Testing accuracy	0.886	0.890	0.926

Step 4. On the last step of models learning assume that 50% of cases when the device 2 fails, and the maneuverable light signal is "blue" (prohibitive), an accident occurs. The remaining data dependencies are not explicit.

	Neural networks	QUEST	C&R Tree
Training accuracy	0.932	0.949	0.874
Testing accuracy	0.928	0.947	0.849

As a result of 4-step models training on available data set, the algorithm showed the best result is QUEST. That's why It is selected for export in the Streams application as a scoring operator.

D. *Railroad situation analysis system parallelism*

*A*fter predictive model is selected and trained it should be translated into InfoSphere Streams application as a user-defined scoring operator for real-time processing of exceedingly large sets of raw data. This raw data is expected to come from various sources such as cameras, RFID sensors, GPS sensors, etc. and be assimilated by Streams. Then initial data will be filtered and divided into several sets each for specific kind of situations to be predicted. This is algorithmic parallelism therefore dividing into 3 parts is referred to as algorithmic decomposition. On this stage of research only one kind of emergency is considered. Since the supposed volume of raw data is enormous, even after dividing it into 3 parts we will still have the bottleneck problem as the size of data for the particular situation can still be too large to meet the target of timely response. In order to avoid this problem data decomposition need to be applied - data need to be split into smaller parts depending on the part of the railroad after that it will proceed to the scoring operators distributed among different processing nodes. Each kind of emergency of course requires it's own set of features, therefore there must be different predictive model (scoring operator) for each one. Thus we have that data parallelism is nested in algorithmic parallelism.

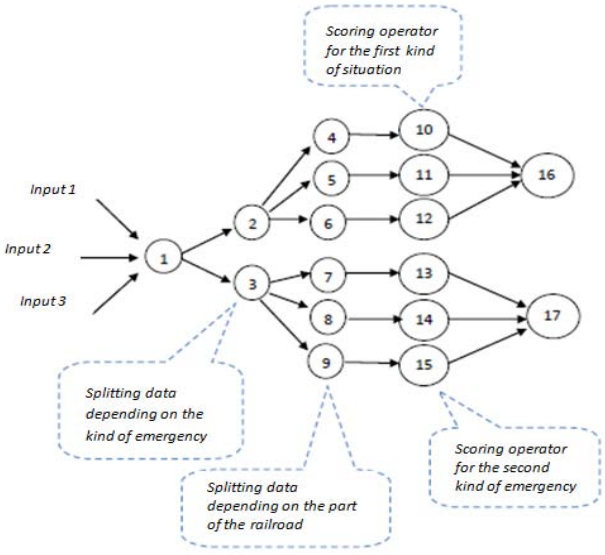


Fig. 2 Railroad situation analysis system parallelism

Parallelism pattern in Infosphere Streams application can be implemented using special operator *Split*. The Split Operator is used to split an input stream into various output streams. These output streams typically route each tuple based on Attribute characteristics.

V. DEVELOPMENT TOOLS

The predictive model will be elaborated with the use of the special data mining software IBM SPSS Modeler.

For parallelization will be used IBM InfoSphere Streams.IBM InfoSphere Streams is a software platform that intended for the development and execution of applications, processing data streams in parallel [5].

The platform provides:

- Streams Processing Language (SPL) consisted of a *programming language* interface that enables end- users operating on data streams and *runtime framework* that can execute the applications on a single or distributed set of hosts in parallel. Streams runtime implements its own message-based interprocess communication model [6] but enables to create applications without needing to understand the lower-level stream-specific operations.
- An integrated development environment (IDE) for Streams applications. Integrating SPSS Model Scoring in InfoSphere Streams makes possible leveraging the powerful predictive models in a real-time scoring environment.

CONCLUSION

According to the research by company DISCOVERY Research Group at the present time in Russia 83% of cargo transportation accounts for the railways. That's why railways security is a key priority for the Russian Government for many years ahead [7]. However current technologies are unable to support predictive detection and prevention of emergencies on railways due to extra large volumes of data and due to the lack of technical means for intellectual data mining in real time. The research conducted by IBM along with NRU-HSE within the state contract is targeted to facilitate russian intellectual rail transport system development.

REFERENCES

[1] Karpenko, A. *Parallel computing.* Retrieved January 29, 2012, from Bauman's MSTU educational base website: http://bigor.bmstu.ru

[2]Wikipedia, Free Encyclopedia, *Mean absolute percentage error.* Retrieved March 3, 2012, from Wikipedia website: http://en.wikipedia.org/wiki/Mean_absolute_percentage_error

[3] Liu, Ling & Ozsu, M. Tamer (Eds.) (2009). Cross Validation. *Encyclopedia of Database Systems*

[4] McLachlan, Geoffrey J. & Do, Kim-Anh & Ambroise, Christophe (2004). *Analyzing microarray gene expression data.* Wiley.

[5] Ballard, C. & Farrell, D. & Lee, M. & Stone P. & Thibault, S. & Tucker, S. (2010). *IBMInfosphere Streams: harnessing data in motion* (pp. 128-130). Texas: IBM International Technical Support Organization.

[6] IBM Corporation. *Transport options.* Retrieved March 25, 2012, from IBM InfoSphere Streams Information Center: http://publib.boulder.ibmcom/infocenter/streams/v2r0/index.j sp

[7] Avdoshin, S. & Gorbatovskiy, M. & Chernov, A. (2011). The concept of the of situational response and modern russian railways safety intellectual system. *Business-informatics*, 4(18), 8-15.