

SYRCoSE 2015

Editors:

Alexander Kamkin, Alexander Petrenko and
Andrey Terekhov

Preliminary Proceedings of the 9th Spring/Summer Young Researchers'
Colloquium on Software Engineering

Samara, May 28-30, 2015

Preliminary Proceedings of the 9th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2015), May 28-30, 2015 – Samara, Russia.

The issue contains selected papers that have been accepted for presentation at the 9th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2015) held in Samara, Russia on May 28-30, 2015. The paper selection was based on originality and contributions to the field. Each paper was peer-reviewed by at least three referees.

The colloquium's topics include programming technologies, formal methods, software and hardware verification, databases and information systems, software safety and security, computer networks and others.

The authors of the selected papers will be invited to participate in a special issue of *The Proceedings of ISP RAS* (<http://www.ispras.ru/proceedings/>), a peer-reviewed journal included into the list of periodicals recommended for publishing doctoral research results by the Higher Attestation Commission of the Ministry of Science and Education of the Russian Federation.

Contents

Foreword	5
Committees	6
Referees	7
FRIS Language Service for Extended Fortran Support in Microsoft Visual Studio <i>I. Ratkevich</i>	8
Pitfalls of C# Generics and Their Solution Using Concepts <i>J. Belyakova, S. Mikhalkovich</i>	17
Visual Parallel Programming as PaaS cloud service with Graph-Symbolic Programming Technology <i>D. Egorova, V. Zhidchenko</i>	24
Procedures Classification for Optimizing Strategy Assignment <i>O. Chetverina</i>	28
Towards Finding Several Bugs at Once by CEGAR <i>V. Mordan, V. Mutilin</i>	34
Acceleration of Profile Creation for Three-Dimensional Vector Video with GPGPU <i>A. Tsyganov</i>	41
Two-Step Harmonious Melody Generator <i>S. Latkina</i>	45
A Crowdsourcing Engine for Mechanized Labor <i>D. Ustalov</i>	52
On the Implementation of a Formal Method for Verification of Scalable Cache Coherent Systems <i>V. Burenkov</i>	56
A Model-Based Approach to Design Test Oracles for Memory Subsystems of Multicore Multiprocessors <i>A. Kamkin, M. Petrochenkov</i>	62
An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms <i>A. Kamkin, A. Protsenko, A. Tatarnikov</i>	66
An Approach to Direct Memory Access Module Verification <i>A. Meshkov, M. Ryzhov, P. Frolov</i>	71
An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs <i>I. Melnichenko, A. Kamkin, S. Smolov</i>	75
Remote Service of System Calls in Microkernel Hypervisor <i>K. Mallachiev, N. Pakulin</i>	82
Constructing Private Service with CRYP2CHAT Application <i>A. Kiryantsev, I. Stefanova</i>	87

Searching Method of Personal Details on the Basis of Fuzzy Comparison <i>N. Limanova, M. Sedov</i>	93
Seamless Development Applicability: an Experiment <i>A. Naumchev</i>	99
Intelligent Design of Class Structure Model based on Ontological Data Analysis <i>A. Kovartsev, V. Smirnov, S. Smirnov</i>	105
Method of Symbolic Test Scenarios Automated Concretization <i>N. Voinov, P. Drobintsev, A. Veselov, V. Kotlyarov, A. Kolchin</i>	109
Unified Model for Testing Object-Oriented Application Development Tools <i>P. Oleynik</i>	112
The Application of Coloured Petri Nets to Verification of Distributed Systems Specified by Message Sequence Charts <i>S. Chernenok, V. Nepomniaschy</i>	119
Carassius: A Simple Process Model Editor <i>N. Nikitina, A. Mitsyuk</i>	129
Iskra: A Tool for Process Model Repair <i>I. Shugurov, A. Mitsyuk</i>	137
Comparing Process Models in the BPMN 2.0 XML Format <i>S. Ivanov, A. Kalenkova</i>	144
Developing of a Complex of Software Tools for Organization and Support of Distance Learning Game System «3Ducation» <i>L. Zelenko, V. Ivanov, A. Grigoriev, A. Semenov, M. Savachaev, E. Poberezkin, D. Konopelkin</i>	149
Combined Classifier for Website Messages Filtration <i>V. Tarasov, E. Mezenceva, D. Karbaev</i>	154
Statistical Data Handling Program of Wireshark Analyzer and Traffic Incoming Research <i>V. Tarasov, G. Gorelov, S. Malakhov</i>	159
Automatic Virtual Link Configuration for Simple AFDX Networks <i>A. Yalaletdinov, A. Khoroshilov</i>	165
Effective Use of Cloud Computing Resources in the Distributed Information Systems for Providing Quality Multimedia Services <i>D. Parfenov, I. Bolodurina</i>	168

Foreword

Dear participants, we are glad to meet you at the 9th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE). The event is held in Samara, a major scientific and educational center of the Volga area. The colloquium is hosted by Povolzhskiy (Volga Region) State University of Telecommunications and Informatics (PSUTI), one of the most known technical schools in Russian Federation. SYRCoSE 2015 is organized by Institute for System Programming of the Russian Academy of Sciences (ISPRAS) and Saint Petersburg State University (SPbSU) jointly with PSUTI.

In this year, Program Committee (consisting of more than 50 members from more than 25 organizations) has selected 29 papers. Each submitted paper has been reviewed independently by three referees. Participants of SYRCoSE 2015 represent well-known universities, research institutes and companies such as A.P. Ershov Institute of Informatics Systems of SB of RAS, Aston, Bauman Moscow State Technical University, Everest Research, INEUM, Information System Technology, Innopolis University, Institute for the Control of Complex Systems of RAS, ISPRAS, Ivanovo State University of Chemistry and Technology, Kazan Federal University, MCST, N.N. Krasovskii Institute of Mathematics and Mechanics of UB of RAS, National Research University – Higher School of Economics, Nazarbayev University, Novaya Platforma, Platov South-Russian State Polytechnic University, PSUTI, Saint-Petersburg State Polytechnic University, Samara State Aerospace University, Samara State Technical University, SKB Kontur, Southern Federal University, SPbSU, The All-Russian Research Institute of Experimental Physics, Ulyanovsk State Technical University, Ural Federal University, V.M. Glushkov Institute of Cybernetics of NAS of Ukraine, VERIMAG Laboratory, Yaroslavl Demidov State University (4 countries, 16 cities and 30 organizations).

We would like to thank all of the participants of SYRCoSE 2015 and their advisors for interesting papers. We are also very grateful to the PC members and the external referees for their hard work on reviewing the papers and selecting the program. Our thanks go to the invited speakers, Susanne Graf (VERIMAG Laboratory), Nikolay Pakulin (ISPRAS) and Nikolay Shilov (Nazarbayev University). We would also like to thank our sponsors and supporters: Russian Foundation for Basic Research (grant 15-07-20201), Exactpro Systems and CyberLeninka. Finally, our special thanks to local organizers, Veniamin Tarasov and Nadezhda Bahareva (PSUTI), for their invaluable help in organizing the colloquium in Samara.


Sincerely yours,

Alexander Kamkin, Alexander Petrenko and Andrey Terekhov
May 2015

Committees

Program Committee Chairs

 Alexander K. Petrenko – Russia
Institute for System Programming of RAS

 Andrey N. Terekhov – Russia
Saint-Petersburg State University

Program Committee

 Jean-Michel Adam – France
Pierre Mendès France University

 Sergey M. Avdoshin – Russia
Higher School of Economics

 Eduard A. Babkin – Russia
Higher School of Economics

 Nadezhda F. Bahareva – Russia
Povolzhskiy State University of Telecommunications and Informatics

 Svetlana I. Chuprina – Russia
Perm State National Research University

 Pavel D. Drobintsev – Russia
Saint-Petersburg State Polytechnic University

 Liliya Yu. Emaletdinova – Russia
Kazan National Research Technical University

 Victor P. Gergel – Russia
Lobachevsky State University of Nizhny Novgorod

 Efim M. Grinkrug – Russia
Higher School of Economics

 Maxim L. Gromov – Russia
Tomsk State University

 Vladimir I. Hahanov – Ukraine
Kharkov National University of Radioelectronics

 Shihong Huang – USA
Florida Atlantic University

 Iosif L. Itkin – Russia
Exactpro Systems

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

 Vsevolod P. Kotlyarov – Russia
Saint-Petersburg State Polytechnic University

 Alexander N. Kovartsev – Russia
Samara State Aerospace University

 Vladimir P. Kozyrev – Russia
National Research Nuclear University "MEPhI"

 Daniel S. Kurushin – Russia
State National Research Polytechnic University of Perm

 Peter G. Larsen – Denmark
Aarhus University

 Roustam H. Latypov – Russia
Kazan Federal University

 Alexander A. Letichevsky – Ukraine
Glushkov Institute of Cybernetics, NAS

 Nataliya I. Limanova – Russia
Povolzhskiy State University of Telecommunications and Informatics

 Alexander V. Lipanov – Ukraine
Kharkov National University of Radioelectronics


 Irina A. Lomazova – Russia
Higher School of Economics


 Lyudmila N. Lyadova – Russia
Higher School of Economics


 Victor M. Malyshko – Russia
Moscow State University

 Vladimir A. Makarov – Russia
Yaroslav-the-Wise Novgorod State University


 Tiziana Margaria – Germany
University of Potsdam

 Marek Miłosz – Poland
Institute of Computer Science, Lublin University of Technology


 Igor A. Minakov – Russia
Institute for the Control of Complex Systems of RAS

 Alexey M. Namestnikov – Russia
Ulyanovsk State Technical University


 Valery A. Nepomniaschy – Russia
Ershov Institute of Informatics Systems of SB of RAS


 Mykola S. Nikitchenko – Ukraine
Kyiv National Taras Shevchenko University

 Yuri S. Okulovsky – Russia
Ural Federal University

 Sergey P. Orlov – Russia
Samara State Technical University


 Elena A. Pavlova – Russia
Microsoft


 Ivan I. Piletski – Belorussia
Belarusian State University of Informatics and Radioelectronics

 Vladimir Yu. Popov – Russia
Ural Federal University

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


 Rustam A. Sabitov – Russia
Kazan National Research Technical University

 Nikolay V. Shilov – Russia
Nazarbayev University

 Ruslan L. Smelyansky – Russia
Moscow State University

 Valeriy A. Sokolov – Russia
Yaroslavl Demidov State University


 Petr I. Sosnin – Russia
Ulyanovsk State Technical University

 Veniamin N. Tarasov – Russia
Povolzhskiy State University of Telecommunications and Informatics

 Sergey M. Ustinov – Russia
Saint-Petersburg State Polytechnic University

 Vladimir V. Voevodin – Russia
Research Computing Center of Moscow State University


 Dmitry Yu. Volkanov – Russia
Moscow State University


 Mikhail V. Volkov – Russia
Ural Federal University

 Nadezhda G. Yarushkina – Russia
Ulyanovsk State Technical University

 Rostislav Yavorsky – Russia
Higher School of Economics

 Nina V. Yevtushenko – Russia
Tomsk State University

 Vladimir A. Zakharov – Russia
Moscow State University

 Sergey S. Zaydullin – Russia
Kazan National Research Technical University

Organizing Committee Chairs and Secretaries

 Alexander K. Petrenko – Russia
Institute for System Programming of RAS

 Nadezhda F. Bahareva – Russia
Povolzhskiy State University of Telecommunications and Informatics

 Veniamin N. Tarasov – Russia
Povolzhskiy State University of Telecommunications and Informatics

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

Referees

Vitaly Antonenko

Eduard Babkin

Nadezhda Bahareva

Mikhail Chupilko

Pavel Drobintsev

Anton Ermakov

Victor Gergel

Nikolay Glazyrin

Efim Grinkrug

Victor Grishchenko

Maxim Gromov

Shihong Huang

Iosif Itkin

Alexander Kamkin

Vsevolod Kotlyarov

Artem Kotsynyak

Alexander Kovartsev

Vladimir Kozyrev

Peter Gorm Larsen

Roustam Latypov

Nataliya Limanova

Nataliya Limanova

Irina Lomazova

Lyudmila Lyadova

Victor Malyshko

Tiziana Margaria

Igor Minakov

Alexey Namestnikov

Alexandr Naumchev

Valery Nepomniaschy

Mykola Nikitchenko

Sergey Orlov

Elena Pavlova

Ivan Piletski

Svetlana Prokopenko

Delhibabu Radhakrishnan

Yury Rogozov

Marco Scavuzzo

Natalia Shabaldina

Nikolay Shilov

Sergey Smolov

Valeriy Sokolov

Petr Sosnin

Alexey Stankevichus

Veniamin Tarasov

Andrei Tatarnikov

Alexander Tchitchigin

Andrei Tiugashev

Pavel Vdovin

Dmitry Volkanov

Mikhail Volkov

Nadezhda Yarushkina

Rostislav Yavorskiy

Nina Yevtushenko

Vladimir Zakharov

FRIS language service for extended Fortran support in Microsoft Visual Studio

Irina Ratkevich

The All-Russian Research Institute of Experimental Physics (RFNC - VNIIEF)
Sarov, Nizhny Novgorod Region,
607188, Russian Federation
E-mail: ratkevichis@gmail.com

Abstract—This report deals with the construction of the language service for extended support of the Fortran programming language in the integrated development environment (IDE) Microsoft Visual Studio. The model and general approach for language service construction is offered. The report focuses on the organization of this model, and the proof of its operability, that is given on the example of the FRIS language service developed by author. The material could be equally applied for construction language services both for other programming languages and for other development environments.

Keywords—*FRIS; Fortran Intelligent Solutions; Fortran; Visual Studio Extensibility; Language Service; Visual Studio*

I. INTRODUCTION

Fortran [1], [2] is one of the first high-level programming languages. It was created in the 50s of XX century and it was intended for development of programs for scientific calculations. Fortran is still used by its intended purpose in the development of simulation programs. Nowadays the most widespread Fortran standard is Fortran 2003 [2] (however there is the Fortran 2008 standard, and the Fortran 2015 standard is in development stage). It cardinally differs from previous standards because it introduces the support of object-oriented programming in a Fortran language. This feature changes the language syntax, where many new statements are added in conjunction with new conceptions. Definitely, such modernizations are necessary, but at the same time they are objectively making the language more complicated.

However these difficulties may be hidden or even eliminated, if the Fortran-programmer will have appropriate assistance from the IDE in which he writes his programs code. The most widely used IDE on Windows is Microsoft Visual Studio. It is extensible and allows adding practically any feature into it. As an example, Visual Studio may be extended to support various programming languages.

The most widely used Visual Studio integrations of the Fortran language are being developed in Intel [3] and PGI [4] in conjunction with corresponding compilers. However the supported features of those integrations significantly inferior to integrations developed by Microsoft, e.g. for C# programming language. Primarily it applies to the support of

IntelliSense [5] technology, which consists of the following features: List Members, Parameter Info, Quick Info и Complete Word (table I).

TABLE I. THE INTELLISENSE TECHNOLOGY FEATURES IMPLEMENTATION IN INTEL AND PGI

Function	Intel	PGI
List Members	No	No
Parameter Info	Yes, excluding overloaded procedures and type bound procedures	Yes, only for intrinsic procedures
Quick Info	Yes, excluding fields and procedures of derived types	Yes, only for intrinsic procedures
Complete Word	Yes, only for modules names, functions names and subroutines names	Yes, only for keywords statements

It must be noted that in all implemented IntelliSense features, excluding those for intrinsic procedures, there is essentially absent any description of the elements except for their definitions.

This great difference between Fortran support and support for languages, developed by Microsoft, became a key factor for author in the decision to implement the FRIS (Fortran Intelligent Solutions) language service, that is intended to cover this gap and implement all IntelliSense features to support Fortran-programmer in effective development of programs.

II. MAKING MODEL OF A LANGUAGE SERVICE

Language service [6] is responsible for providing language-specific support for editing source code in the Visual Studio IDE, or, generally speaking, in any IDE. Basic language service must by definition [7] to provide a program syntax highlighting, all other features, including the IntelliSense support, are extra (or extended) features. The main question that must be answered at first when starting a new language service development is what features are needed for a programmer. After that, those features must be ranked by priority (or by usability).

Next, it is needed to identify the sources of data that must be used in the implementation of the language service. The main data source for any language service, no doubt, is source

files containing programs text on a target language, but in some cases additional data sources may be needed.

The next stage is to estimate implementation complexity of needed functions. This estimation may include as the IDE restrictions to different components of a language service, and the analysis complexity of the target programming language itself.

After this the aggregate language service model is constructed, that reflects its major structural elements and interconnections between them. This report contains generalized and optimal, in author's opinion, language service model, which provides extended support for a target programming language.

When the aggregate language service model is constructed, each of its structural elements is detailed according to specific requirements to implementation of different features, and also depending on the restrictions of the target programming language.

Next in the report each of aforementioned steps in making language service will be examined in details, on example of the Fortran programming language, but the given material, without loss of generality, could be applied to any other programming language.

A. Analysis of requirements and the necessary features

The first thing, that definitely wants to see any programmer is a program syntax highlighting, for keywords, data type names, string literals, comments and so on. At the same time, it's important to provide the ability to configure such highlighting, for example, for significant to user procedure names and data type names of program libraries, say, OpenMP, MPI, and others. Such syntax highlighting helps to focus attention on the most important details.

The second thing, that is important to a programmer, is the amount of provided context help, that at least must consist of the definition for a programming language element with which programmer works or wants to work (in the case of word completion lists). But in most cases the element definition is not enough to understand, how exactly the element must be used, as an example, a procedure that has more than a dozen parameters, some of which may be optional. In this case it's necessary to accompany the element definition with some meaningful description. When the data that must be provided to user, and, respectively, that must be collected and stored, are identified, the sources, from which this could be obtained, must be analyzed.

B. Analysis of data sources

The most obvious way to get the definitions of programming language elements is the analysis of program source files. The form of such definitions is fixed in the programming language standard, e.g. in the Fortran standard. The meaningful description of the elements may be obtained, if to complement the program text with comments in a special form – documentation comments. The XML documentation comments are the standard for Visual Studio. So, the program

text contains two languages: the base language – Fortran, and the embedded language – documentation comments language.

It should be noted, that Fortran has a distinctive feature in using of the programming libraries. There are three ways to connect the programming library to the main Fortran project:

- with source code files, that contains the library API, including procedure definitions, data types definitions and so on;
- with compiled binary files of Fortran modules, that have a closed format, which understandable just by compiler. Those files also contains the library API definitions;
- without any descriptions of library API. In such case the compiler will deduce the outer interfaces for used procedures, and will try to resolve external references by their names.

In the first case, it is possible to analyze source file that contains the library API and get all necessary information from it, but in the other two cases, it's impossible to do so, and it's necessary to provide other mechanisms to get such information.

As a basis for implementation of this task, was taken the idea that is used in the program for automatic documentation generation for so called managed applications – Sandcastle [8]. It uses two files for generation of program documentation: one with the API description, and the other with the documentation for the API.

Fortran isn't managed language, so it's impossible to use the standard Sandcastle API format for description of its elements. Therefore the model for description Fortran API was developed in FRIS for this purpose. It is the XML file in the special format, which contains a description of main Fortran elements. FRIS can save (serialization) the structure of elements, which is obtained from the analysis of program texts, into XML format and restore (deserialization) Fortran elements from their XML representation.

The XML model for Fortran documentation comments is also developed, including the features for its serialization and deserialization. This will allow to develop a special Sandcastle plug-in, and to use files of Fortran API and documentation comments description to automatically generate a developer or/user help files.

C. Analysis of main operating characteristics of a language service

When developing a language service it's necessary to take into account that analysis of program texts will operate in a real time. This means that in most cases the text under analysis will be in the lexical, syntactic or semantic incorrect state, in terms of programming language specification. This peculiarity must be considered in the construction of corresponding analyzers.

The second peculiarity is in the fact that the analysis for a syntax highlighting is carried out in Visual Studio line-by-line (one line a time). The analyzer, colorizer in terms of VS, is

transmitted for analysis a string of text and the analyzer state in which it was at the end of analysis of the previous line. This means that the corresponding analyzer must be constructed with the ability to save its state in any time and to restore its work from any such state. This approach makes it possible to carry out incremental analysis, which is very important for large source files (approx more than 10000 lines). Then, when some lines are changed, it's necessary to analyze just the changed lines, but not a whole file.

The third peculiarity that must be considered to create effective full-text analyzers is the need to take into account the state of source files. In terms of using program project source files in the IDE, file could be in a one of two essential states:

- opened in editor;
- doesn't opened in editor.

In the first case, it's needed to accomplish full-text analysis of a source files, but in the second one it's possible to accomplish a simplified analysis to collect information about just externally visible program elements. For example, it's not necessary to analyze whole body of procedure, because information, say, about its local variables could be needed to user just in a moment of editing a procedure body, which automatically transfers file with procedure to the state "opened in editor", and consequently, the other analysis rules will be applied to it. Thus the requirement to analyzer to operate in two modes, for convenience "full" and "simplified" analysis, will significantly increase the analysis speed of programming project source files.

III. GENERAL MODEL OF A LANGUAGE SERVICE

The author proposes the following general model for building any language services, which is the result of summarizing author's experience in developing FRIS (Fig. 1).

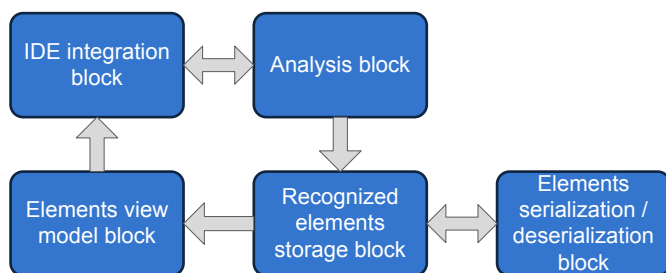


Fig. 1. General language service model

As shown in Fig.1 any language service could be represented as 5 base blocks. The arrows represent the data exchange between blocks.

The IDE integration block contains interfaces implementation, which are required for interaction with IDE. It's responsible for subscription of a language service on the text editing editor events, and for corresponding responses, for example, for syntax highlighting and information providing for work of IntelliSense features.

The analysis block is responsible for lexical, syntactic and semantic analysis. When it receives events from the IDE

integration block, it performs appropriate actions. For example, in response to file open event or text changed event, it will provide the information for syntax highlighting. It's also responsible for providing source files analysis depending on their states.

The recognized elements storage block is central data storage about all elements, necessary for language service. In general case, it is kind of a symbol table. The storage block could be filled from two sources: from analysis block, as a result of analysis of a source files, and from serialization/deserialization block, in the case of using model of Fortran API for any program libraries.

The elements serialization/deserialization block performs two functions. Firstly, it allows saving the content of programming projects as XML files for description of Fortran API and documentation comments. Secondary, it allows restoring the content of programming projects from their XML models. This approach reflects the dual nature of programming projects. Thus, for author of programming project, for example, program library, it is accessible in source files and it is perceived as "internal", but for a user of this library, it is perceived as "external", and its source files may be inaccessible to user.

The elements view model block is a link, a kind of adaptor for elements of storage block to their representation needed by IDE integration block. Thus, recognized elements may contain some information that is not necessary to IntelliSense technology features, or on the contrary, does not contain some needed information. The elements view model is playing this interconnection role. It contains data types that are wrappers for elements of storage block, which fulfils requirements of the IDE integration block. There is also implemented various functions of filtering and selecting of different kinds necessary information. It could be said, that the storage block is like a database, and the view model block is like a data selection procedures.

A. IDE integration block

The IDE integration block connects a language service with a basic IDE infrastructure. In the case of Visual Studio, the base language service must implement the IVsLanguageInfo [9] interface. This interface is responsible for providing information about target language including its name, associated file extensions, and component for a syntax highlighting (colorizer). Colorizer must to implement the IVsColorizer [10] interface, which is responsible for providing character-by-character information about colors of buffered program text representation in memory. In order to provide the IntelliSense technology support it is needed to implement 5 additional interfaces [11]: IVsCodeWindowManager, IVsMethodData, IVsCompletionSet, IVsTextViewFilter and IOleCommandTarget.

To simplify for developers the task of creating new language services, and the other tasks of Visual Studio extension, Microsoft created MPF (Managed Package Framework) [12] library, which supplies a set of base classes that implements many needed interfaces, and thus provides to developers the ability to implement only the features that is

needed to them. Let's take a brief look at the key classes that are necessary for the implementation of the language service and its various features.

The LanugageService abstract class provides basic implementation of a language service. It contains a number of abstract methods responsible for different features of a language service, such as syntax highlighting, and initialization of full-text source files analysis in order to provide information for various IntelliSense features, and so on.

The Source class is a source file abstraction in terms of a language service. It is used to store all information about edited file, as well as for interoperability with other language service model classes, which require information about current source file. In particular, it contains an instance of the Colorizer class, which is responsible for syntax highlighting.

The Colorizer class implements IVsColorizer interface. This class is used by the core editor of IDE for providing of syntax highlighting in current source file. For even more flexibility and abstraction MPF Colorizer from concrete programming language, the scanner abstraction is used.

The scanner must to implement IScanner interface. Each scanner is essentially a specialized lexical analyzer, which must be able to save its current state and to restore its state for continuation of analysis as if it is doing a simple linear analysis of character stream.

The AuthoringScope class contains all information about a source file which is the result of parsing of this file. It is the central place for providing information for basic IntelliSense technology features. In particular, method GetDataTipText – returns a string that contains description of programming language element, under the mouse cursor. It provides data for Quick Info IntelliSense feature. Method GetDeclarations – returns a list of programming language element definitions. It provides data for List Members and Complete Word IntelliSense features. Method GetMethods – returns a list of method signatures with a given name, including their overloaded versions. It provides data for Parameter Info IntelliSense feature.

In FRIS implementation is used modified version of MPF library, since a number of methods needed by FRIS were inaccessible for overriding in Microsoft's MPF classes.

B. Analysis block

The FRIS analysis block consists of two sub blocks: analysis for syntax highlighting and full-text analysis (in "full" and "simplified" mode) for a collection of information about elements in a source file.

The FRIS analyzers are built with the ability to support sublanguages. In this case, the base language is Fortran, and sublanguages are any other languages, other than Fortran, that are used in the program text, for example, the XML documentation comments language and the OpenMP directives language.

Fig. 2 shows the general scheme of working of the analyzers stack, on the example of analysis of a part of XML

documentation comment. The base language analyzer (Fortran) generates tokens, which are then passed through a tokens filter. If token matches with one of registered sublanguages, the appropriate analyzer is called. The output is a set of fully recognized tokens for all supported languages.

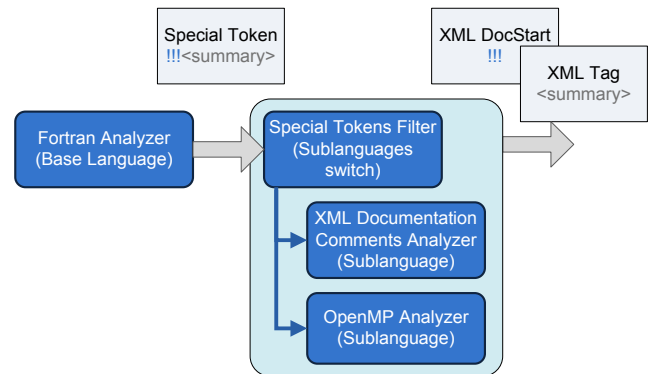


Fig. 2. The general analyzers operation scheme

The peculiarity of work of a syntax highlighting analyzing block is that it is essentially some kind of extended version of a lexical analyzer, since there are strict requirements on the speed of operation of a syntax highlighting. Support for arbitrary program library in FRIS is, in particular, in the ability of a visual highlighting of their elements such as procedures, modules, data types, etc. Such highlighting is performed in a syntax highlighting block based on the current context. For any identifier under analysis the check depending on current scope is performed, whether it belongs to arbitrary library, which elements necessary to highlight. Then, if necessary, the identifier is highlighted with a defined earlier color.

The peculiarity of full-text analysis is in the used analysis strategy. Since the analysis is need to be performed in the real time, while the user modifies the text of program, all analyzers must to work in the error suppression mode. It must be noted that Fortran is very complicated language for analysis, because of its lexical and syntactical peculiarities. The most striking examples are:

- the ability to use multiline tokens, for example, identifiers. Next is given the sample of a multiline identifier "my_id". The special attention must be given the fact that in between a start and end lines of any multiline lexeme, it is allowed to use comments and blank lines.

```

1  my_&
2  !comment
3
4  !another comment after blank line
5  &id
  
```

- the absence of reserved keywords. The decision whether identifier is a keyword depends on a context of its usage in a statement. Therefore, it is not statements that are identified by keywords, as in languages with reserved keywords, but the keywords are identified by statements. Taking into account that analysis is performed in a real time, it is impossible to determine

the identity of incomplete statement. For example, it is unclear, whether “if” is a keyword that belongs to conditional statement, or it is a name of an array, in the following part of statement: “if”.

The emphasized peculiarities greatly complicate the development of analyzers for Fortran. But all of them are taken into account in FRIS. In particular, the optimistic parsing strategy is used. The parser processes a source file statement-by-statement. For every statement the abstract syntax tree (AST) is built. If the statement could not be matched, e.g. as a result of that the user just not has time to completely type it; the special AST is generated for it, which includes all mismatched tokens.

In conjunction with a parser the full AST builder is operating (Fig.3). It builds the full AST from the individual statement ASTs. It also stores the AST that is already built. The builder task is to track operations of opening and closing of syntactical contexts, in particular their optimistic completion.

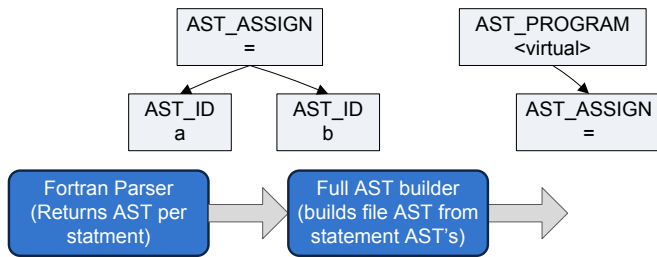


Fig. 3. The FRIS parser operation scheme

For example, if now the operator “if(...)then” is analyzed, then according to standard, it could be completed only by “endif” statement. However, the user could not have enough time to fully type this statement, then the builder will interpret the “end” statement as a completion of a “if(...)then” operator. Similarly to it, if in the end of parsing of source file the stack of open contexts of the builder is not empty, then they are completing in a special mode – completion by the end of the file. It is also have ability of priority processing of high level element statements. For example, if the subroutine element is processed now, and as a result of a parsing the function element definition statement is discovered, then the current subroutine element is being completed with a special flag, and the function element processing is being started.

Thus, the parser is always outputs the correct AST, which has no error nodes. This allows simplifying the semantic analysis algorithm. The semantic analyzer walks the AST and collects information about all needed Fortran elements, which then stores in the recognized elements storage block.

C. The recognized elements storage block

The recognized elements storage block is a central storage for all known in the current programming project elements (modules, data types, variables, etc.). It is filled from two sources: as a result of a source files parsing, and as a result of deserializing information about arbitrary libraries.

This block is essentially a kind of a symbol table. Its design must take into account that information in it will be continuously updating as a result of the user editing of source files.

Consider the proposed generic model of the storage block (Fig. 4).

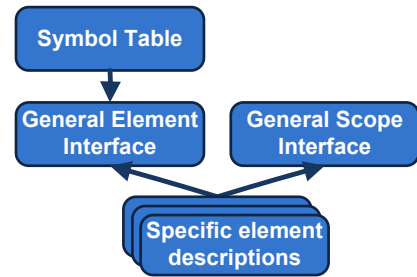


Fig. 4. The model of the recognized elements storage block

It consists of following parts:

- the class for a symbol table description;
- the class for an interface description for a typical element of the programming language;
- the class for an interface description for a typical scope of the programming language;
- the classes describing specific elements of the programming language, that implement interfaces of a typical element and of a typical scope, for elements, which are scopes.

The class for a symbol table description must be built as indexed data storage, in order to effectively processing operations of update and elements search. For maximum flexibility it must store the references on the interface for a typical element, instead of references to specific elements. The specific element could be obtained from an abstract interface as a result of type casting. The following scheme of a symbol table is proposed (Table II).

TABLE II. THE MODEL OF A SYMBOL TABLE

Field	Data type	Description
Names	map<long, string>	Map unique identifier to string
Elements	map<long, object>	Map element unique identifier to element object
Projects	map<string, map<string, list<long>>>>	Map program project name to map of project file names to list of file elements unique identifiers
ProjectDependencies	map<string, list<string>>	Map program project to program projects it depends from

In this approach, firstly there is an access to all elements (Elements field). Secondly, for any project there is a list of its dependencies from other projects, which allows simplify a search procedure of needed elements, and to exclude from the search result the elements that is not visible in target project. Thirdly, every project contains a dictionary of its source files, and elements, which contained in every file that allows to effectively performing the update operations. The update operation is a result of a source file parsing operation, due to a

text changes made by user. Thus, since all elements that are connected with file is known, so their deletion from other dictionaries and insertion of a newly recognized elements, is a relatively simple task.

Next consider the proposed interface for a typical element of a programming language (table III).

TABLE III. THE MODEL OF INTERFACE FOR A TYPICAL ELEMENT OF A PROGRAMMING LANGUAGE

Field	Data type	Description
<i>Name</i>	string	Name of element
<i>Scope</i>	Scope	Outer scope of element
<i>Description</i>	string	Description of element. For instance from documentation comments
<i>Location</i>	Location	Element location: definition location, declaration location. Location consists of file name and region. Region consists of 4 integer indexes: start line, start line character index, end line, end line character index.

Every element must have at least a name, a scope, where it's defined, a description, for example, that is obtained from documentation comments, and a location. An element location consists from a declaration location and a definition location. Each of which is in turn consists from a file name, and an element region in it.

Consider the proposed interface for a typical scope of the programming language (table IV). The scope, in a general case, is a container of elements.

TABLE IV. THE MODEL OF A TYPICAL SCOPE OF THE PROGRAMMING LANGUAGE

Field	Data type	Description
<i>Scope</i>	Scope	Outer scope of this scope
<i>Elements</i>	list<Element>	List of elements of the scope

Every scope contains a reference to a parent scope and a list of elements that make up this scope.

Every specific element of a programming language must be derived from an interface for a typical element, and if it is a scope, from an interface of a typical scope.

D. The elements serialization/deserialization block

The elements serialization/deserialization block is a key element for the implementation of a mechanism to support arbitrary user libraries. The serialization mechanism performs a saving of a given programming project in a form of two special XML files: description of Fortran API and description of documentation comments. The optional level of refinement could be additionally specified. In the case, when the serialization is performed for creation a developer documentation of a programming project, then all elements are saved, but in the case of creation a user documentation or interface for a programming project as an external library, then just externally visible elements are saved. It should be recalled that for each element in the Fortran module, could be specified the access mode: public or private. The public elements are externally accessible when the module is used, but the private elements could be used just inside the module and inaccessible outside of it.

The deserialization mechanism operation is slightly different, because in deserialization there is just one operation mode – reading all information describing an arbitrary library. In this case, even if there will be provided XML files, that contains full description of arbitrary library, only externally visible elements will be read. This allows reducing the amount of memory needed to store a library description, and also eliminates the need to store elements, which will not be accessed to user under no circumstances, for example, private module elements, or internal elements of procedures.

For serialization and deserialization are used the models for description of Fortran API and XML documentation comments, that is developed by author and are expressed in the form of appropriate XML Schema Definitions (XSD) [13], [14]. Let's consider each of these models.

The model of Fortran API (Fig.5) allows describing external interfaces of any library as a Fortran interfaces. The meaning and purpose some of the model elements are given in table V.

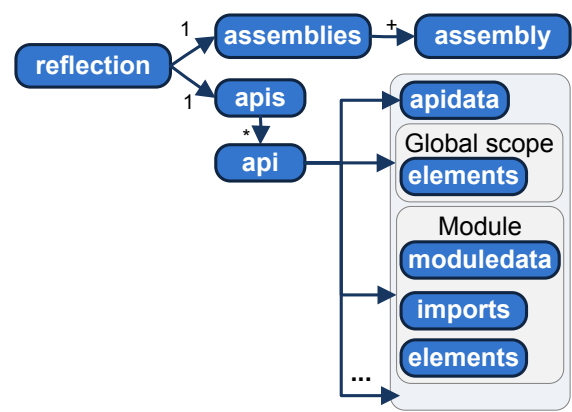


Fig. 5. The part of Fortran API XSD

TABLE V. THE DESCRIPTION OF SOME ELEMENTS OF THE FORTRAN API MODEL

Element (tag)	Description
<i>reflection</i>	Root tag
<i>assemblies</i>	Describes set of projects that API contained in this file
<i>assembly</i>	Describes individual project
<i>apis</i>	Root for all API description
<i>api</i>	Element description
<i>apidata</i>	Describes group and subgroup of element. I.e. for function: group – method, subgroup - function
<i>moduledata</i>	Module description switch
<i>referencedata</i>	Reference element switch
<i>typedata</i>	Derived type description switch
<i>variabledata</i>	Variable description switch
<i>proceduredata</i>	Procedure description switch
<i>interfacedata</i>	Interface description switch
<i>methoddata</i>	Method description switch
<i>namelistdata</i>	Name list description switch
<i>commonblockdata</i>	Common block description switch
<i>imports</i>	Module imports description
<i>elements</i>	List of inner elements

As can be seen from the above figure, tag “apis” contains a description of all project elements. The tag “api” is used for a direct element description. In order to uniquely identify the

type of element: a module, a function, a subroutine, a data type and so on, the special switches, like a “moduledata” tag, are used.

One more remark should be made regarding the tag “elements”, which is used to describe the internal elements of current element. It’s allowed to specify here references – fully qualified element names, and their description place next in a main “apis” tag, and also it’s allowed to provide the description of child elements directly in this tag.

It should be noted that description of Fortran API may be used for a creation of Fortran procedure interfaces for their calls from other programming languages, that is solves the inverse problem.

Consider the model of documentation comments. It conceptually consists of two interconnected parts: a description of documentation tags for documenting program elements (Fig.6), and a description of documentation comments XML file format (Fig. 7). The meaning and purpose of the model elements are given in table VI.

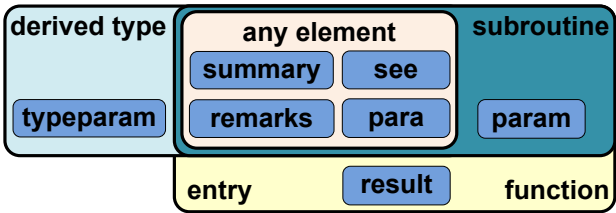


Fig. 6. The usage of documentation tags for different Fortran elements

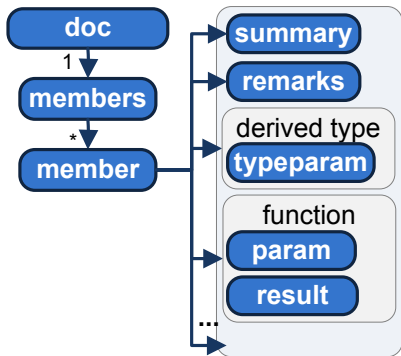


Fig. 7. The part of Fortran Documentation XSD

TABLE VI. THE ELEMENTS DESCRIPTION OF THE FORTRAN DOCUMENTATION MODEL

Element (tag)	Description
<i>doc</i>	Root element
<i>members</i>	Container for all documentation elements
<i>member</i>	Contains documentation for single element
<i>summary</i>	Element summary
<i>remarks</i>	Additional information for element
<i>see</i>	Internal tag, makes reference to given element
<i>para</i>	Internal tag, creates paragraph in parent tag
<i>typeparam</i>	Describes derived type parameter
<i>param</i>	Describes argument of subroutine or function
<i>result</i>	Describes function result

For description of any element may be used 4 tags, two of which are high-level: “summary” and “remarks”, and other

two are nested, it means that they could be used just inside of other tags: “see” and “para”. In addition to them, for description of:

- derived type parameters is used “typeparam” tag;
- arguments of subroutines, functions and entry points is used “param” tag;
- result of function is used “result” tag.

Thus, files for description of the model of Fortran API and documentation comments form the basis not only for work with arbitrary libraries in Fortran, but also form the basis for the generation of the reference documentation, for example with a Sandcastle tool. It should be noted that Fortran API model can be used for solving the inverse problem – description of API for a Fortran procedures for their using from other programming languages.

E. The elements view model block

The elements view model block is a link between the IDE integration block and the data storage block. It performs two basic functions: converts a data from a storage block to a form required by the IDE, and performs various search operations in a storage block.

The convert operation of stored data to the form required by the IDE produces elements that are complemented by the properties of visual representation. For example, such properties as text color and element icon, which used in various completion lists, are set. In other words, the elements view model block contains various aspects of data presentation to user. Thus the structure of the view model block is analogue to the structure of the storage block. It also defines interfaces for typical presentation elements and scopes, and a set of their specific implementations for each element of the storage block.

The second function of this block is the search function. Here are performed various operations of elements resolution in a scope, a search for elements with the specified name and type, etc. That is, it performs the selection of needed elements from the storage block that taking into account a different aspects of a programming language. Then, selected data converted to the form required for user representation.

IV. PROOF OF CONCEPT

The FRIS language service is built on the basis of the general model of a language service, and implements all described blocks. Figures 8-13 are examples of work of its various functions, proving the presented conception of a generalized language service model, including providing extended support for user libraries.

```

type(UrsOfData) ofdata
call ReleaseUrsOf(ofdata,ko,kan)

↓

type(UrsOfData) ofdata
call ReleaseUrsOf(ofdata,ko,kan)

```

Fig. 8. The extended support of user libraries (before and after)

```
class(extendedtype) :: ex
call ex%
```

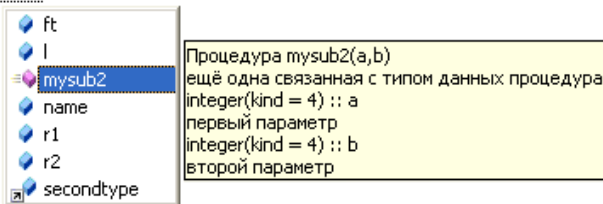


Fig. 9. List Members

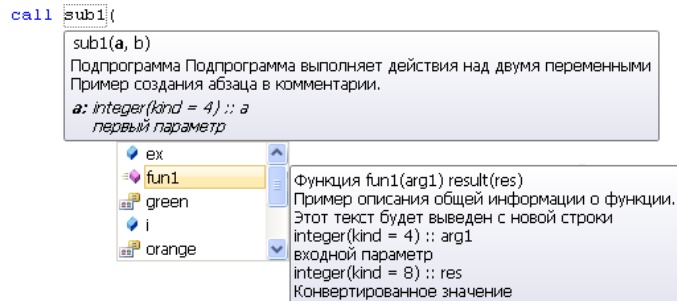


Fig. 10. Parameter Info and Complete Word

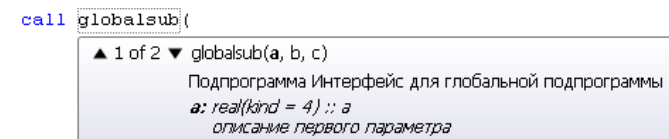


Fig. 11. Parameter Info for overloaded subroutine

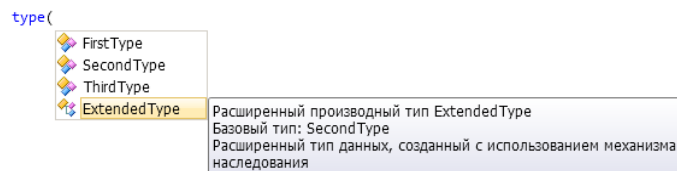


Fig. 12. Complete word for a derived type name

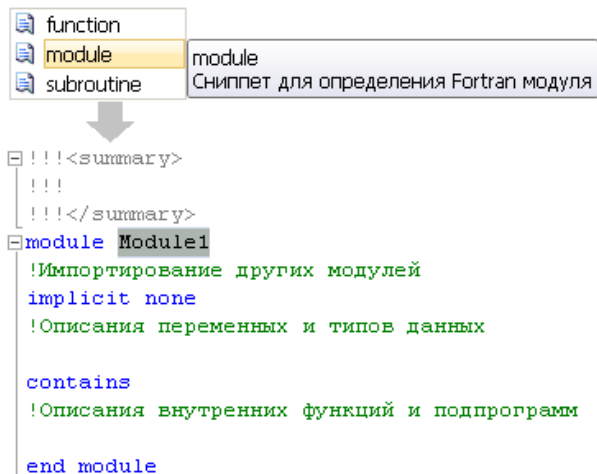


Fig. 13. Code Snippet Sample

Consider the pivot table of the language services from Intel, PGI and FRIS (table VII).

TABLE VII. THE INTEL, PGI AND FRIS LANGUAGE SERVICES COMPARISON

Function	Intel	PGI	FRIS
List Members	No	No	Yes
Parameter Info	Yes, excluding overloaded procedures and type bound procedures	Yes, only for intrinsic procedures	Yes
Quick Info	Yes, excluding fields and procedures of derived types	Yes, only for intrinsic procedures	Yes
Complete Word	Yes, only for modules names, functions names and subroutines names	Yes, only for keywords statements	Yes
Code Snippet [15] Support	Yes, but only as menu command or shortcut	No	Yes. Snippets included in Completion Lists
Documentation comments support	No	No	Yes. Documentation included in all tooltips
Support of user libraries	No	No	Yes

Thus, due to use of the developed general language service model, FRIS provides extended support of a Fortran in Microsoft Visual Studio.

V. CONCLUSION

The report presents the general model of a language service for extended support of a Fortran programming language developed by author. This model can be easily applied not only to create new language services for other languages, but also to create a language services in other IDEs.

All aspects that must be taken into account in development of a language service are given in details, including the analysis of user requirements, the analysis of a data sources for a language service, and the analysis of operation peculiarities of a language service in a specific IDE.

As a result of executing described analysis kinds, in every particular case, the plan of a language service development must be created. For a language service development simplification, the general model of a language service is given and each its block is described in details on example of its implementation in FRIS.

At last, the proof of proposed concept of constructing language services is given, on example of comparison FRIS with existing language services from Intel and PGI. The model that is used in FRIS provides its significant advantage over other language services.

It especially should be noted that FRIS implements a model for supporting user libraries. It includes a model of Fortran API and a model of documentation comments, developed by author. The Fortran API model allows not only to describe the interfaces of any library in terms of Fortran,

but also allows solving the inverse problem, by known Fortran interfaces obtain API for target language. The documentation comments model allows user to document different Fortran elements straight in the program text, and then obtain documentation in various types of context help. The model of Fortran API in conjunction with the model of documentation comments can be used to create a developer and/or user documentation, for example with a Sandcastle tool.

REFERENCES

- [1] The Fortran automatic coding system for the IBM 704 EDPM. Programmers reference manual. IBM, 1956
- [2] ISO. ISO/IEC 1539-1:2004 Information technology - Programming languages - Fortran -Part 1: Base Language, pp. 569
- [3] Intel Fortran Composer (Visual Fortran) URL: <http://software.intel.com/en-us/articles/intel-fortran-composer-xe-2013-sp1-release-notes>
- [4] PGI Visual Fortran URL: <https://www.pgroup.com/products/pvf.htm>
- [5] Using IntelliSense URL: [http://msdn.microsoft.com/en-us/library/hcw1s69b\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/hcw1s69b(v=vs.80).aspx)
- [6] Language Services URL: <http://msdn.microsoft.com/en-us/library/bb165099.aspx>
- [7] Model of a Language Service URL: [http://msdn.microsoft.com/en-us/library/bb166518\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/bb166518(v=vs.100).aspx)
- [8] Eric Woodruff's Sandcastle Help File Builder Documentation URL: <http://ewsoftware.github.io/SHFB/html/bd1ddb51-1c4f-434f-bb1a-ce2135d3a909.htm>
- [9] IVsLanguageInfo Interface URL: [https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.textmanager.interop.ivslanguageinfo\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.textmanager.interop.ivslanguageinfo(v=vs.80).aspx)
- [10] IVsColorizer Interface URL: [https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.textmanager.interop.ivscolorizer\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/microsoft.visualstudio.textmanager.interop.ivscolorizer(v=vs.80).aspx)
- [11] Language Service Interfaces URL: [http://msdn.microsoft.com/en-us/library/bb164598\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/bb164598(v=vs.80).aspx)
- [12] Managed Package Framework Classes URL: [http://msdn.microsoft.com/en-us/library/bb164709\(v=vs.80\).aspx](http://msdn.microsoft.com/en-us/library/bb164709(v=vs.80).aspx)
- [13] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures URL: <http://www.w3.org/TR/xmlschema11-1/>
- [14] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes URL: <http://www.w3.org/TR/xmlschema11-2/>
- [15] Creating and Using IntelliSense Code Snippets URL: [https://msdn.microsoft.com/en-us/library/ms165392\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/ms165392(v=vs.80).aspx)

Pitfalls of C# Generics and Their Solution Using Concepts

Julia Belyakova

Institute for Mathematics, Mechanics
and Computer Science
Southern Federal University
Rostov-on-Don, Russia
Email: julbel@sfned.ru

Stanislav Mikhalkovich

Institute for Mathematics, Mechanics
and Computer Science
Southern Federal University
Rostov-on-Don, Russia
Email: miks@sfned.ru

Abstract—In comparison with Haskell type classes and C++ concepts, such object-oriented languages as C# and Java provide much limited mechanisms of generic programming based on F-bounded polymorphism. Main pitfalls of C# generics are considered in this paper. Extending C# language with *concepts* which can be simultaneously used with interfaces is proposed to solve the problems of generics; a design and translation of concepts are outlined.

I. INTRODUCTION

Generic programming is supported in different programming languages by various techniques such as C++ templates, C# and Java generics, Haskell type classes, etc. Some of these techniques were found more expressive and suitable for generic programming, other ones more verbose and worse maintainable [1]. Thus, for example, the mechanism of expressive and flexible C++ unconstrained templates suffers from unclear error messages and a late stage of error detection [2], [3]. New language construct called **concepts**¹ was proposed for C++ language as a possible substitution of unconstrained templates. A design of C++ concepts² conforms to main principles of effective generic tools design [1].

In comparison with concepts and Haskell type classes [1], [7], such mainstream object-oriented languages as C# and Java provide much limited mechanisms of generic programming based on F-bounded polymorphism. Pitfalls of C# generics are analysed in this paper in detail (Sec.II): we discuss some known drawbacks and state the problems of subtle semantics of recursive constraints (Sec.II-B) and constraints-compatibility (Sec.II-C). To manage the pitfalls considered extending of C# with concepts is proposed: a design of concepts is briefly presented in Sec.IV. We also discuss a translation of such extension to standard C#.

C# language is used in this paper primarily for the sake of syntax demonstration. As for the pitfalls of C# generics, they hold for Java as well with slight differences. However, while the concepts *design* proposed in the paper could be

¹ Term “concept” was initially introduced in a documentation of the Standard Template Library (STL) [4] to describe requirements on template parameters in informal way.

²There were several designs of C++ concepts [3], [5], [6]; all of them share some general ideas.

```
(ICmp-1) interface IComparable<T> {int CompareTo(T other);}
(ICmp-2) interface IComparer<T>    {int Compare(T x, T y);}

(s-1)    Sort<T>(T[]) where T : IComparable<T>;
(s-2)    Sort<T>(T[], IComparer<T>);
```

Fig. 1. IComparable<T>/IComparer<T> interfaces and its applications

easily adapted for Java (and also for any .NET-language with interface-based generics), the technique of language extension *translation* (which we consider in Sec.IV) cannot be applied for Java directly. Unlike Java Virtual Machine, .NET Framework preserves type information in its byte code, this property being crucial for the translation method.

II. PITFALLS OF C# GENERICS

C# and Java interfaces originally developed to be an entity of object-oriented programming were later applied to generic programming as constraints on generic type parameters. There are several shortcomings of this approach.

A. Lack of Retroactive Interface Implementation

Interfaces cannot be implemented *retroactively*, i.e. it is impossible to add the relationship “type T implements interface I” if type T is already defined. Consider a generic algorithm for sorting arrays Sort<T> with the following signature:

```
Sort<T>(T[]) where T : IComparable<T>;
```

If some type Foo provides an operation of comparison but does not implement the interface IComparable<Foo>, Sort<Foo> is not a valid instance of Sort<>. What one can do in this case? If type cannot be changed (it may be defined in external .dll, for instance), the only way to cope with sorting is to define an adapter class FooAdapter which implements Sort<FooAdapter> interface, pack all Foo objects into FooAdapter ones, sort them and unpack back to an array of Foo objects. Apparently, there must be a better approach.

Fortunately, in the .NET Framework standard library the Array.Sort<T> method [8] is provided with two “branches” of overloads:

- 1) For any type T which implements IComparable<T> interface ((s-1) example, Fig. 1).

```
(1) interface IComparableTo<S> { int CompareTo(S other); }
(2) interface IComparable<T> where T : IComparable<T>
    { int CompareTo(T other); }
```

Fig. 2. IComparable<T> vs IComparableTo<S> example

- 2) For any type T with an external comparer of type $IComparer<T>$ provided ((s-2) example, Fig. 1).

Hence, if some type is already defined, values of this type can be compared, but this type does not implement `IComparable<>` interface (as in the `Foo` example above), `Sort<>` with `IComparer<>` (branch 2) is to be used. Thus one can simulate retroactive modeling property (in Scala the similar approach is referred to as a programming with the “concept pattern” [9]). Consequently, if retroactive modeling is required, a programmer has to write a generic code twice — in “interface-oriented” and in “concept pattern” styles. The amount of necessary overloads grows exponentially: if one needs two retroactively modeled constraints on generic type, corresponding generic code would consist of four “twins”, if three — eight “twins” and so on.

B. Drawbacks of Recursive Constraints

Example 1. The following reason about the `Sort<T>` method for `IComparable<T>` may be not obvious. The notation of `Sort<T>` in (s-1) example (Fig. 1) looks a little bit redundant; such a recursive constraint on type T might look even frightening, but it is well formed. Furthermore, the word “comparable” in this context is very likely associated with the ability to compare values of type T with each other. But the interface `IComparable<T>` ((ICmp-1), Fig. 1) does not correspond this semantics: it designates the ability of some type (which implements this interface) to be comparable with type T . The same problem with `Comparable<X>` interface in Java is explored in [10]. The particular role of recursive constraints in generic programming is explored in [11].

It would be better to split the single `IComparable<>` interface into two different interfaces (Fig. 2):

- 1) `IComparableTo<S>` which requires some type (which implements this interface) to be comparable with S .
- 2) `IComparable<T>` which requires values of type T to be comparable with each other.

Note that the definition of the latter interface needs the constraint `where T : IComparable<T>` (q.v. Fig. 2).

Example 2. As an another example consider a generic definition of graph with peculiar structure: graph stores some data in vertices; every vertex contains information about its predecessors and successors thereby defining arcs. A graph itself consists of set of vertices instead of set of edges. Such kind of graph is suitable for a task of data flow analysis in the area of optimizing compilers [12] because “movement along arcs up and down” is intensively used action in an analysis of a control flow graph.

Fig. 3 illustrates parts of the corresponding definitions: `IDataGraph<Vertex, DataType>` describes interface of a data graph; `IDataVertex<Vertex, DataType>` describes interface

```
interface IDataVertex<Vertex, DataType>
    where Vertex : IDataVertex<Vertex, DataType> // (*)
{
    ...
    IEnumerator<Vertex> OutVertices { get; }
    ...
}
interface IDataGraph<Vertex, DataType>
    where Vertex : IDataVertex<Vertex, DataType> // (#)
{
    ...
}
```

Fig. 3. IDataGraph<, > and IDataVertex<, > interfaces

```
static HashSet<T> GetUnion<T>(HashSet<T> s1, HashSet<T> s2)
{
    var us = new HashSet<T>(s1, s1.Comparer);
    us.UnionWith(s2);
    return us;
}
```

Fig. 4. Union of HashSet<T> objects

of a vertex in such graph. While the graph interface really depends on type parameters `Vertex` and `DataType`, we have to include `Vertex` as a type parameter into the vertex interface `IDataVertex<, >` as well. Similarly to `IComparable<>` example the constraints (*) and (#) in Fig. 3 are not superfluous. Suppose we have the following types:

```
class V1 : IDataVertex<V1, int> { ... }
class V2 : IDataVertex<V1, int> { ... }
```

Thanks to the constraints (*) and (#) the instantiation of graph `IDataGraph<V2, int>` is not allowed, since type `V2` does not implement interface `IDataVertex<V2, int>`. Without these constraints we might accept some inconsistent graph with vertices of type `V2` which refer to vertices of type `V1`.

Vertex and graph interface definitions are unclear and non-obvious. If programmers might be used to use interface `IComparable<>`, it is more difficult to manage such things as `IDataGraph<, >` example. In some cases one may prefer to abandon writing generic code because of this awkwardness.

C. Ambiguous Semantics of Generic Types

When using flexible `Sort<T>` method with an external `IComparer<T>` parameter (Fig. 1), a programmer has clear understanding of how elements are sorted, since such a comparer is a *parameter of an algorithm*. But when one uses generic *types*, this information is implicit. For instance, `SortedSet<T>` class takes `IComparer<T>` object as a constructor parameter, `HashSet<T>` class taking `IEqualityComparer<T>`. Therefore, given two sets of the same generic type one cannot check at *compile time* whether these sets are *constraints-compatible* (in case of `HashSet<T>` “constraints-compatibility” means that the given sets use the same equality comparer). And it seems that a programmer usually does not suppose that objects of the same type can have different comparers (or addition operators, coercions, etc). But they can, and it leads to subtle errors.

Suppose we have a simple function `GetUnion<T>` (q.v. Fig. 4) which returns a union of the two given sets. If some arguments `a` and `b` provide different equality comparers (e.g., case-sensitive and case-insensitive comparers for type `string`), the result of `GetUnion(a, b)` would differ from the result of `GetUnion(b, a)`. Note that Haskell type classes do not suffer

```

interface IObservable<O, S> where O : IObservable<O, S>
    where S : ISubject<O, S>
{
    void update(S subj);
}

interface ISubject<O, S> where O : IObservable<O, S>
    where S : ISubject<O, S>
{
    List<O> getObservers();
    void register(O obs);
    void notify();
}

```

Fig. 5. Observer pattern in C#

from such an ambiguity because every type provides only one instance of a type class.

D. The Problem of Multi-Type Constraints

The well-known problem of multi-type constraints holds for C# interfaces. Requirements concerning on several types cannot be naturally expressed within interfaces. The paper [10] deals with the example of Observer pattern in Java. The Observer pattern connects two types: Observer and Subject. Both types has methods which take the another type of this pair as an argument: the Observer provides `update(Subject)`, the Subject — `register(Observer)`.

Fig. 5 shows the interface definitions `IObservable<O, S>` for Observer and `ISubject<O, S>` for Subject in standard C#. We need two different interfaces and have to duplicate the constraints on `O` and `S` in both definitions to establish consistent connection between type parameters `O` and `S`. And again we face with recursive constraints on types `O` (which represents the Observer) and `S` (which represents the Subject). This example looks even worse than the case of vertex and graph interfaces presented in Fig. 3. But it is the only way to define a type family [13] of Observer pattern correctly.

E. Constraints Duplication and Verbose Type Parameters

All constraints required by a definition of generic type are to be repeatedly specified in every generic component which uses this type. Consider the generic algorithm `GetSubgraph<G, >` depending on type parameter `G` which implements `IDataGraph<Vertex, DataType>` interface (q.v. Fig. 3).

```

G GetSubgraph<G, Vertex, DataType>(
    G g, Predicate<DataType> p)
where G : IDataGraph<Vertex, DataType>, new()
where Vertex : IDataVertex<Vertex, DataType> { ... }

```

`GetSubgraph<G, Vertex, DataType>` method is not correct without explicit specification of constraint on type parameter `Vertex`. This constraint is induced by the definition of `IDataGraph<Vertex, DataType>` interface and should be repeated every time one uses `IDataGraph<G, >`.

Another property of `GetSubgraph<...>` definition is a plenty of generic parameters. Clearly, vertex and data types are fully determined by the type of specific graph. At the level of `GetSubgraph<...>` signature vertex type even does not matter at all. Such types are often referred to as *associated types*. Some programming languages allow to declare associated types explicitly (SML, C++ via traits, Scala via abstract types and some other), but in C# and Java they can only be represented by extra type parameters. It makes generic definitions verbose and breaks encapsulation of constraints on

associated types. Issues of repeated constraints specification and lack of associated types are considered in [14], [1] in more detail.

III. RELATED WORK

We consider two studies concerning modification of generic interfaces in this section:

- 1) [14] proposes the extension of C# generics with associated types and constraint propagation.
- 2) [10] generalizes Java 1.5 interfaces enabling retroactive interface implementation, multi-headed interfaces (expressing multi-type constraints) and some other features.

Both studies *revise interfaces* to improve interface-based mechanism of generic programming and to approach to C++ concepts and Haskell type classes, which are considered being rather similar [7]. Some features of Scala language in respect to problems considered in Sec. II will also be mentioned.

A. C# with Associated Types and Constraint Propagation

Member types in interfaces and classes are introduced in [14] to provide direct support of *associated types*. A mechanism of *constraint propagation* is also proposed to lower verbosity of generic components and get rid of constraints duplication as was mentioned in Sec. II-E. The example of Incidence Graph concept from the Boost Graph Library (BGL) [15] is considered. It is shown that features proposed can significantly improve a support of generic programming not only in C# language but in any object-oriented language with F-bounded polymorphism.

But the problems of multi-type constraints and recursive constraints cannot be solved with this extension. Thus, the code of Observer pattern (Fig. 5) cannot be improved at all because of recursive constraints; the same holds for `IComparable<T>` interface. The issue of retroactive implementation is also not touched upon in [14]: extended interfaces are still interfaces which cannot be implemented retroactively.

B. JavaGI: Java with Generalized Interfaces

In contrast to [14], the study [10] is mainly concentrated on the problems of retroactive implementation, multi-type constraints (solved with *multi-headed interfaces*) and recursive interface definitions³. For instance, Observer pattern is expressed in JavaGI with generalized interfaces as shown in Fig. 6 [10]. Methods of a whole interface are grouped by a receiver type with keyword `receiver`. A syntax of an interface looks a little bit verbose but it is essentially better than two interfaces with duplicated constraints shown in Fig. 5. Moreover, JavaGI interfaces allow *default implementation* of methods (as `register` and `notify`). Retroactive implementation of interfaces is also allowed, but it is possible to define only one implementation of an interface for the given set of types in a namespace.

³This problem is usually connected with so-called *binary methods problem*.

```

interface ObserverPattern[S, O] {
  receiver O { void update(S subj); }
  receiver S {
    List<O> getObservers();
    void register(O obs) { getObservers().add(obs); }
    void notify() { ... }
  }
}
class MultiheadedTest {
  <S,O> void genericUpdate(S subject, O observer)
    where [S,O] implements ObserverPattern {
    observer.update(subject);
  }
}

```

Fig. 6. Observer pattern in JavaGI

It turns out that interfaces become some restricted version of C++ concepts [5], [16] (in particular, they do not support associated types) and, moreover, they lose a semantics of object-oriented interfaces⁴. JavaGI interfaces only act as *constraints* on generic type parameters, but they cannot act as types, so one cannot use JavaGI interfaces as in Java.

C. “Concept Pattern” and Context Bounds in Scala

The idea of programming with “concept pattern” has been reflected in Scala language [9]. Due to the combination of generic *traits* (something like interfaces with abstract types and implementation), *implicit*s (objects used by default as function arguments or class fields) and *context bounds* (like $T : Ordering$ in Fig. 7) Scala provides much more powerful mechanism of generic programming than C# or Java. Fig. 7 illustrates the examples of sorting and observer pattern.

Context bounds provide simple syntax for single-parameter constraints: the sugared (s-s) version of `Sort[T]` algorithm is translated into (s-u) one by desugaring. Retroactive modeling is supported since one can define new `Ordering[]` object and use it for sorting. And one does not need to provide two versions of the sort algorithm as for C# language (q.v. Fig. 1): `Sort[]` with one argument would use default ordering due to `implicit` keyword. `ObserverPattern[S, O]` looks rather similar to corresponding JavaGI interface (Fig. 6). There is no syntactic sugar for multi-parameters traits, so the notation of `genericUpdate[S, O]` cannot be shortened.

In respect to the *constraints-compatibility* problem discussed in Sec. II-C Scala’s “concept pattern” reveals the same drawback as C#. Generic types take “concept objects” as constructor parameters. In such a way `TreeSet[A]` [17] implicitly takes `Ordering[A]` object, therefore, for instance, the result of intersection operation would depend on an order of arguments if they use different ordering.

IV. DESIGN OF CONCEPTS FOR C# LANGUAGE

A. Interfaces and Concepts

It seems that a *new language construct* for generic programming should be introduced into such object-oriented languages as C# or Java. If we extend interfaces preserving their object-oriented essence [14], a generic programming mechanism becomes better but still not good enough, since such problems as

⁴The way to preserve compatibility with Java code is considered in [10], but “real interfaces” no longer exist in JavaGI.

```

(s-s) def Sort[T : Ordering](elems: Array[T]) { ... }
(s-u) def Sort[T](elems: Array[T])
      (implicit ord: Ordering[T]) { ... }

trait ObserverPattern[S, O] {
  def update(obs: O, subj: S);
  def getObservers(subj: S): Seq[O];
  def setObservers(subj: S, observers: Seq[O]);
  def register(subj: S, obs: O)
    { setObservers(subj, getObservers(subj) ++ obs); }
  def notify(subj: S) { ... }
}

object MultiheadedTest {
  def genericUpdate[S, O](subject: S, observer: O)
    (implicit obsPat: ObserverPattern[S, O]) {
    obsPat.update(observer, subject);
  }
}

```

Fig. 7. `Sort[T]` and `ObserverPattern[S,O]` examples in Scala

retroactive modeling or constraints-compatibility remain. If we make interfaces considerably better for generic programming purposes [10], they lose their object-oriented essence and can no longer be used as types.

We advocate the assertion that *both* features have to be provided in an object-oriented language:

- 1) Object-oriented **interfaces** which are used as *types*.
- 2) Some new construct which is used to *constrain* generic type parameters. C++ like **concepts** are proposed to serve this goal.

B. C# with Concepts: Design and Translation

In this section we present a sketch of C# concepts design. Concept mechanism introduces the following constructs into the programming language:

- 1) **Concept**. Concepts describe a named set of *requirements* (or *constraints*) on one or more types called *concept parameters*.
- 2) **Model**. Models determine the manner in which specific types *satisfy* concept. Models are external for types; they can be defined later than types. It means that a type can *retroactively* model a concept if it semantically conforms to this concept. Types may have several models for the same concept. In some cases a default model can be implicitly generated by a compiler.
- 3) **Constraints** are used in generic code to describe requirements on generic type parameters.

Concepts support the following kinds of constraints:

- associated types and associated values;
- function signatures (may have default implementation);
- nested concept requirements (for concept parameters and associated types);
- same-type constraints;
- subtype and supertype constraints;
- aliases for types and nested concept requirements.

The main distinction of C# concepts proposed in comparison with other concepts designs (C++, G [16]) is the support of *subtype* constraints and *anonymous models* (like anonymous classes). Concept-based mechanism of constraining generic type parameters surpasses the abilities of interface-based one.

Construct of extended language	Construct of base language
Concept	Abstract class
Concept parameter	Type parameter
Associated type	Type parameter
Concept refinement	Subtyping
Associated value	Property (only read)
Nested concept requirement	Type parameter
Concept requirement in generic code	Type parameter
Model	Class

Fig. 8. Translation of C# extension with concepts

At the same time interfaces can be used as usual without any restrictions.

Concepts can be implemented in existing compilers via the translation to standard C#. Fig. 8 presents correspondence between main constructs of extended and standard C# languages. To preserve maximum information about the source code semantics, some additional metainformation has to be included into translated code. In particular, one needs to distinguish generic type parameters in the resultant code as far as they may represent concept parameters, associated types or nested concept requirements. To resolve such ambiguities we propose using *attributes*.

The method of translation suggested is strongly determined by the properties of .NET Framework. Due to preserving type information and attributes in a .NET byte code, translated code can be unambiguously recognized as a result of code-with-concepts translation. Moreover, it can be restored into its source form, what means that *modularity* could be provided: having the binary module with definitions in extended language one can add it to the project (in extended language either) and use in an ordinary way.

Fig. 9 illustrates several concept definitions (in the left column) and their translation to standard C# (in the right column). Basic syntax of concepts is shown: concept declarations (start with keyword `concept`), signature constraints, signature constraints with default implementation (`NotEqual` in `CEquatable[T]`), refinement (`concept CComparable[T] refines CEquatable[T]`, i.e. it includes all requirements of refined concept and adds some new ones), associated types (`Data in CTransferFunction[TF]`), multi-type concept `CObserverPattern[O, S]`, nested concept requirements (`CSemilattice[Data] in CTransferFunction[TF]`).

Concepts are translated to generic classes. Function signatures are translated to abstract or virtual (if implementation is provided) class methods. Concept parameters and associated types are represented by type parameters (marked with attributes) of a generic abstract class as well as *nested concept requirements*. For instance, `CSemilattice_Data` type parameter of `CTransferFunction<>` denotes `CSemilattice[Data]` concept requirement because this parameter is attributed with `[IsNestedConceptReq]`, corresponding subtype constraint being in a where-clause.

Some examples of generic code with concept constraints are presented in the left column of Fig. 10. Concept requirements can be used with alias (as `CComparable[T]` in the class of binary search tree). Note that a singular definition of generic component is sufficient. Translated generic code (in the right

```
static bool Contains<T>(T x, IEnumerable<T> values)
    where CEquatable[T] { ... }
static void TestContains
{
    Rational[] nums = ...;
    var hasNumber5 = Contains[model CEquatable[Rational]] {
        bool Equal(Rational x, Rational y)
        { return x.Num == y.Num; }
    } (new Rational(5), nums);
}
```

Fig. 12. Anonymous model example

Feature	G	C++	C# ^{ext}	JGI	ScI	C# ^{cpt}
multi-type constraints	+	+	± ¹	+	+ ²	+
associated types	+	+	+	—	+	+
same-type constraints	+	+	+	—	+	+
subtype constraints	—	—	+	+	+	+
retroactive modeling	+	+	± ¹	+	+ ³	+
multiple models	+	—	± ¹	—	+	+
anonymous models	—	—	—	—	+ ³	+
concept-based overloading	+	+	—	—	± ⁴	—
constraints-compatibility	+	+	—	+	—	+

“C#^{ext}” means C# with associated types [1].

“ScI” means Scala [9].

“C#^{cpt}” means C# with concepts.

¹partially supported via “concept pattern”

²supported via “concept pattern”

³supported via “concept pattern” and implicits

⁴partially supported by prioritized overlapping implicits

Fig. 13. Comparison of “concepts” designs

column) demonstrates significant property of translation: concept requirements are translated into extra type parameters instead of extra method and constructor parameters (as it is in Scala and G [16]). Therefore, constraints-compatibility can be checked at compile time, methods and objects being saved from unnecessary arguments and fields.

Fig. 11 presents the model of concept `CComparable[]` for class `Rational` of rational number. It is translated to derived class `CComparable_Rational_Def` of `CComparable<Rational>` and then used as the second type argument of generic instance `BST<, >`. Fig. 12 demonstrates using of anonymous model to find a number with a numerator equal to 5.

V. CONCLUSION AND FUTURE WORK

Many problems of C# and Java generics seem to be well understood now. Investigating generics and several approaches to revising OO interfaces, we faced with some pitfalls of these solutions which were not considered yet.

- 1) Recursive constraints used to solve the binary method problem appear to be rather complex and often do not correspond a semantics assumed by a programmer.
- 2) The “concept pattern” breaks constraints-compatibility.
- 3) Using interfaces both as types and constraints on generic type parameters leads to awkward programs with low understandability.

To solve problems considered we proposed to extend C# language with the *new* language construct — **concepts**. Keeping interfaces untouched, concept mechanism provides much better support of the features crucial for generic programming [1]. The support of these features in C# with concepts

<pre> concept CEquatable[T] { bool Equal(T x, T y); // function signature // function signature with default implementation bool NotEqual(T x, T y) { return !Equal(x, y); } } // refining concept concept CComparable[T] refines CEquatable[T] { int Compare(T x, T y); // overrides Equal from refined concept CEquatable[T] override bool Equal(T x, T y) { ... } } concept CTransferFunction[TF] { type Data; // associated type // nested concept requirement require CSemilattice[Data]; Data Apply(TF trFun, Data d); TF Compose(TF trFun1, TF trFun2); } concept CObserverPattern[O, S] { void UpdateSubject(O obs, S subj); ICollection<O> GetObservers(S subj); void RegisterObserver(S subj, O obs) { GetObservers(subj).Add(obs); } void NotifyObservers(S subj) { ... } } </pre>	<pre> [Concept] abstract class CEquatable<[IsConceptParam]T> { public abstract bool Equal(T x, T y); public virtual bool NotEqual(T x, T y) { return !this.Equal(x, y); } } [Concept] abstract class CComparable<[IsConceptParam]T> : CEquatable<T> { public abstract int Compare(T x, T y); public override bool Equal(T x, T y) { ... } } [Concept] abstract class CTransferFunction< [IsConceptParam]TF, [IsAssocType]Data, [IsNestedConceptReq]CSemilattice_Data> where CSemilattice_Data : CSemilattice<Data>, new() { public abstract Data Apply(TF trFun, Data d); public abstract TF Compose(TF trFun1, TF trFun2); } [Concept] abstract class CObserverPattern< [IsConceptParam]O, [IsConceptParam]S> { public abstract void UpdateSubject(O obs, S subj); public abstract ICollection<O> GetObservers(S subj); public virtual void RegisterObserver(S subj, O obs) { GetObservers(subj).Add(obs); } public virtual void NotifyObservers(S subj) { ... } } </pre>
---	--

Fig. 9. Concept examples and their translation to basic C#

<pre> static void Sort<T>(T[] values) where CComparable[T] { ... } class BinarySearchTree<T> // concept requirement with alias where CComparable[T] using cCmp { private BinTreeNode<T> root; ... private bool AddAux(T x, ref BinTreeNode<T> root) { ... // reference to concept by alias if (cCmp.Equal(x, root.data)) return false; ... } } </pre>	<pre> [GenericFun] static void Sort<[IsGenericParam]T, [IsRequireConceptParam]CComparable_T>(T[] values) where CComparable_T : CComparable<T>, new() { ... } [GenericClass] [ConceptAlias("CComparable_T", "cCmp")] class BinarySearchTree<[IsGenericParam]T, [IsRequireConceptParam]CComparable_T> where CComparable_T : CComparable<T>, new() { private BinTreeNode<T> root; ... private bool AddAux(T x, ref BinTreeNode<T> root) { ... CComparable_T cCmp = ConceptSingleton<CComparable_T>.Instance; if (cCmp.Equal(x, root.data)) return false; ... } } </pre>
---	---

Fig. 10. Generic code and its translation to basic C#

<pre> // class for rational number with properties // Num for numerator and Denom for denominator class Rational { ... } model CComparable[Rational] { bool Equal(Rational x, Rational y) { return (x.Num == y.Num) && (x.Denom == y.Denom); } int Compare(Rational x, Rational y) { ... } } ... BST<Rational> rations = new BST<Rational>(); // * </pre>	<pre> class Rational { ... } [ExplicitModel] class CComparable_Rational_Def : CComparable<Rational> { public override bool Equal(Rational x, Rational y) { return (x.Num == y.Num) && (x.Denom == y.Denom); } public override int Compare(Rational x, Rational y){...} } ... BST<Rational, CComparable_Rational_Def> rations // * = new BST<Rational, CComparable_Rational_Def>(); </pre>
---	---

* "BST" is used instead of "BinarySearchTree" for short.

Fig. 11. Model CComparable[Rational] and its translation to basic C#

extension and its comparison with some other generic mechanisms are presented in Fig. 13. The design of C# concepts is rather similar to C++ concepts designs, but it supports subtype and supertype constraints.

We also suggested a novel way of concepts translation: in contrast to G concepts [16] and Scala "concept pattern" [9], C# concept requirements are translated to *type* parameters instead of object parameters; this lowers the run-time expenses on passing extra objects to methods and classes.

Much further investigation is to be fulfilled. First of all, type safety of C# concepts has to be formally proved. The

design of concepts proposed seems to be rather expressive, but it needs an approbation. So the next step is developing of the tool for compiling a code in C# with concepts. Currently we are working on formalization of translation from extended language into standard C#.

ACKNOWLEDGMENT

The authors would like to thank the participants of the study group on the foundations of programming languages Vitaly Bragilevsky and Artem Pelenitsyn for discussions on topics of type theory and concepts.

REFERENCES

- [1] R. Garcia, J. Jarvi, A. Lumsdaine, J. Siek, and J. Willcock, “An Extended Comparative Study of Language Support for Generic Programming,” *J. Funct. Program.*, vol. 17, no. 2, pp. 145–205, Mar. 2007.
- [2] B. Stroustrup and G. Dos Reis, “Concepts — Design Choices for Template Argument Checking,” C++ Standards Committee Papers, Technical Report N1522=03-0105, ISO/IEC JTC1/SC22/WG21, October 2003.
- [3] G. Dos Reis and B. Stroustrup, “Specifying c++ concepts,” in *Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL ’06. New York, NY, USA: ACM, 2006, pp. 295–308.
- [4] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.
- [5] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, G. Dos Reis, and A. Lumsdaine, “Concepts: Linguistic Support for Generic Programming in C++,” in *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications*, ser. OOPSLA ’06. New York, NY, USA: ACM, 2006, pp. 291–310.
- [6] B. Stroustrup and A. Sutton, “A Concept Design for the STL,” C++ Standards Committee Papers, Technical Report N3351=12-0041, ISO/IEC JTC1/SC22/WG21, January 2012.
- [7] J.-P. Bernardy, P. Jansson, M. Zalewski, S. Schupp, and A. Priesnitz, “A Comparison of C++ Concepts and Haskell Type Classes,” in *Proceedings of the ACM SIGPLAN Workshop on Generic Programming*, ser. WGP ’08. New York, NY, USA: ACM, 2008, pp. 37–48.
- [8] “System.Array.Sort(T) Method,” URL: <http://msdn.microsoft.com/library/system.array.sort.aspx>.
- [9] B. C. Oliveira, A. Moors, and M. Odersky, “Type Classes As Objects and Implicits,” in *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, ser. OOPSLA ’10. New York, NY, USA: ACM, 2010, pp. 341–360.
- [10] S. Wehr, R. Lammel, and P. Thiemann, “JavaGI: Generalized Interfaces for Java,” in *ECOOP 2007 Object-Oriented Programming*, ser. Lecture Notes in Computer Science, E. Ernst, Ed., vol. 4609. Springer Berlin Heidelberg, 2007, pp. 347–372.
- [11] B. Greenman, F. Muehlboeck, and R. Tate, “Getting F-bounded Polymorphism into Shape,” in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI ’14. New York, NY, USA: ACM, 2014, pp. 89–99.
- [12] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools (2Nd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006, ch. Code Optimization.
- [13] E. Ernst, “Family Polymorphism,” in *Proceedings of the 15th European Conference on Object-Oriented Programming*, ser. ECOOP ’01. London, UK, UK: Springer-Verlag, 2001, pp. 303–326.
- [14] J. Järvi, J. Willcock, and A. Lumsdaine, “Associated Types and Constraint Propagation for Mainstream Object-oriented Generics,” in *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA ’05. New York, NY, USA: ACM, 2005, pp. 1–19.
- [15] *The Boost Graph Library: User Guide and Reference Manual*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [16] J. G. Siek, “A Language for Generic Programming,” Ph.D. dissertation, Indianapolis, IN, USA, 2005, aAI3183499.
- [17] “TreeSet[A] Class,” URL: <http://www.scala-lang.org/api/current/#scala.collection.mutable.TreeSet>.

Visual Parallel Programming as PaaS cloud service with Graph-Symbolic Programming Technology

Darya Egorova

Software Systems Department, Information Science Faculty
Samara State Aerospace University (SSAU)
Samara, Russia

Victor Zhidchenko

Software Systems Department, Information Science Faculty
Samara State Aerospace University (SSAU)
Samara, Russia
vzhidchenko@yandex.ru

Abstract—In this paper we present the visual approach to parallel programming provided by Graph-Symbolic Programming Technology. The basics of this technology are described as well as advantages and disadvantages of visual parallel programming. The technology is being implemented as a PaaS cloud service that provides the tools for creation, validation and execution of parallel programs on cluster systems. The current state of this work is also presented.

Keywords—parallel; programming; visual; graph; tool; cluster; cloud

I. INTRODUCTION

Text is traditionally used for describing computer programs. While programs are sequential, it is convenient to express them as text, because the nature of text is sequential. A sequence of letters comprises a word. A sequence of words comprises a sentence. A sequence of sentences forms a text. An order of letters in a word, an order of words in a sentence and an order of sentences in a text are very important. Changing any of them can substantially change the text, especially when this text describes some computer program.

On the other hand, when a program is parallel, its text representation becomes inconvenient. In parallel program you want to see which parts of a program can run concurrently and sequential text form can not show it. You have to imagine interdependencies between different program parts and guess possible combinations of their concurrent execution. When the program is large you have to scroll it up and down to see the parts which actually can run concurrently.

This is where a graphical representation can help. A graphical or visual form is usually bidirectional, so you can easily distinguish sequential and parallel parts of a program. Another important factor is that visual representation is more suitable for human comprehension than a text. When you want to explain something you often get a piece of paper and begin to draw a scheme. The drawing is usually more explanative than a text, it is more compact and is easier to remember.

There is also a substantial disadvantage in using graphics for parallel programs representation. A parallel program often consists of hundreds or thousands of threads or processes and the actual number of them is may be unknown prior to program's execution. Moreover, the number of threads can

vary during execution. When you write such a program in the text, it can be very compact. The clarity still suffers but due to the compactness it is quite easy to imagine the threads structure. Trying to depict such program graphically leads to more complex representation of it. As you can not display thousands of threads on one picture, you have to replace them with some abstract graphics structure. The clarity suffers as well as in the case of the text. So instead of the intuitively clear picture you get some abstraction which is less compact than text and whose usability depends on the chosen abstract form.

There are many ways the visual means are used in programming. Most of them are auxiliary to the "traditional" text programming as they help to perform some particular tasks like building class diagrams, dependency graphs or trace logs. Natural visual programming is provided by visual programming languages. Most of them represent a program as a graph which consists of nodes connected to each other by some links (directed or undirected). Depending on the meaning of nodes and links there are many different approaches to represent a program which can be split into several sets:

- UML diagrams [1]
- Domain-specific Visual Languages
- Petri Nets
- Finite-state and Automata-based Programming [2]
- Data Flow Diagrams
- Control Flow Diagrams

In this paper we describe the present results of the work carried out during several years in Samara State Aerospace University (SSAU) in developing methods and tools for visual parallel programming. We use as a basis the visual programming technology for sequential programming, which is called Graph-Symbolic Programming Technology (GSP-technology) also developed in SSAU [3]. We have extended this technology to describe parallel programs and have evolved it through several desktop versions to development environment working with computing cluster. Today we are working on migrating this technology to the cloud and making PaaS service for visual parallel programming. The results of our work have been used as methods and tools of parallel

programming in the education process in SSAU and in research activity in the area of numerical analysis.

II. THE BASICS OF GRAPH-SYMBOLIC PROGRAMMING TECHNOLOGY

GSP-technology represents the program as a graph. The nodes of this graph are little programs (modules), which perform simple operations on variables of project domain. The set of variables form a data dictionary.

The nodes are connected with links. The links show the flow of control between the nodes. Every link is provided with the predicate – a logic condition, which permits or denies the flow of control by this link. This condition is a logical function, defined on variables from the data dictionary.

There are situations, when several links going from one node have a true predicate. To resolve this issue, each link has a priority. The link with the highest priority defines the flow of control.

A graph may contain another graph as a node – so, the program is a graph hierarchy. Fig. 1 shows an example graph that solves quadratic equations.

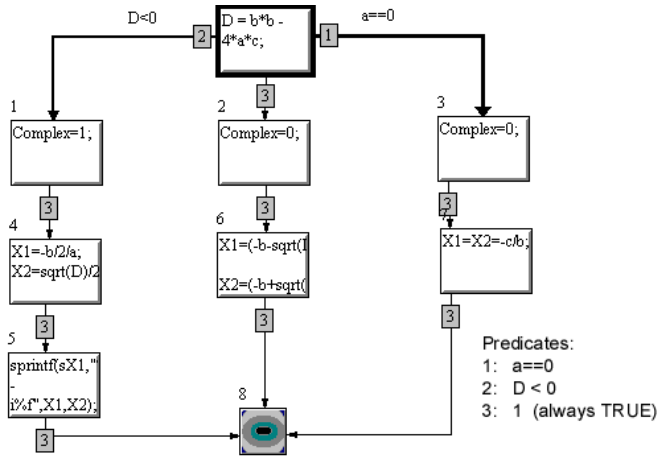


Fig. 1. Graph of a program for solving quadratic equations

The benefits of GSP are:

- Clear and compact representation of the control flow in a program.
- Elimination of many programming errors as graphic representation is very simple for a human and helps to see many logic errors and inconsistencies.
- Simplicity of the program modification.
- Automatic data flow between the nodes. A programmer is protected from making an error on this stage.
- The program structure is stored into a database. It helps to perform many automatic tasks, such as graph structure verification, measuring of graph complexity,

automatic control of graph hierarchy consistency, automatic testing and convenient debugging of programs, automatic creating of program documentation.

Being sequential by default, the GSP-technology was further developed for creating parallel programs. GSP graphic representation of programs helps to solve main parallel programming problems:

- Program's visualization.
- Complexity of the interprocess synchronization.

Many tasks have explicit parallelism. The trivial example is determination of real roots of a quadratic equation. GSP graphic representation is very suitable for such tasks. You can simply draw two (or several) parallel branches instead of thinking how to put in order different tasks and how to represent them in a convenient manner.

The graphic language of GSP-technology is expanded with two types of links:

- The parallel link (a link that shows the beginning of a parallel branch) is labeled with the circle in the beginning.
- The terminating link (a link which determines the end of a parallel branch) is labeled with inclined segment.

The program is divided into several processes, which can be performed in parallel. Each process is represented as a separate branch - a set of nodes interconnected with ordinary links and executed sequentially. The number of branches is unlimited. It is forbidden to connect two nodes from different branches.

All branches operate on the same set of data defined in data dictionary. Sometimes, for the purposes of performance optimization and convenience, it is necessary to define local copies of the same data for each parallel branch. It is accomplished by setting the flag "local" for the corresponding variable in data dictionary. The variables with "local" flag set are created in each process separately during execution.

Synchronization is accomplished with a semaphore technique. A special "synchronization graph" is constructed together with the main program graph. The nodes remain unchanged while the links represent nodes interdependences. A link, drawn from Node₁ to Node₂, means, that Node₂'s execution depends on Node₁'s state. Transmitting of Nodes' state is made by means of messages.

$L_c = [C_{i0,j}^k, C_{i1,j}^k, \dots, C_{im,jr}^k]$ is a Message list, where C_{ij}^k is a message with the number k, sent to Node_i from Node_j.

If L_c contains C_{ij} , then Node_i informs Node_j about the finish of its execution.

Every node checks messages addressed to it, before execution. A special semaphore predicate is evaluated on these messages. In accordance with the previous example:

$R_j = f(C_{i0,j}^k, C_{i1,j}^k, \dots, C_{im,j}^k)$ is a semaphore predicate of Node_j. R_j is a logical function. If $R_j = \text{TRUE}$, then Node_j starts execution, in other case it waits for the truth of R_j .

If all data in a program are independent and there is no need to synchronize parallel branches, the synchronization graph becomes unnecessary and is not built. When it is necessary to synchronize some parts of parallel branches, the user draws synchronization links between the corresponding nodes depicting the sources and targets of synchronization messages. The rest of synchronization graph is implicit and is built automatically.

The process of parallel program development in GSP-technology includes the following steps:

- Data dictionary setup – determining types and variables, needed to solve a problem.
- Modules generation. Modules are written in one of the programming languages (C++ is now supported). They are executed sequentially.
- Drawing the program graph.
- Predicates generation. Predicates are written as boolean functions in the same programming language as modules.
- Drawing the synchronization graph if necessary.
- Semaphore predicates generation for the nodes being synchronized.
- Program compiling and building an executable file.

Fig. 2 shows an example of the graph of the parallel program.

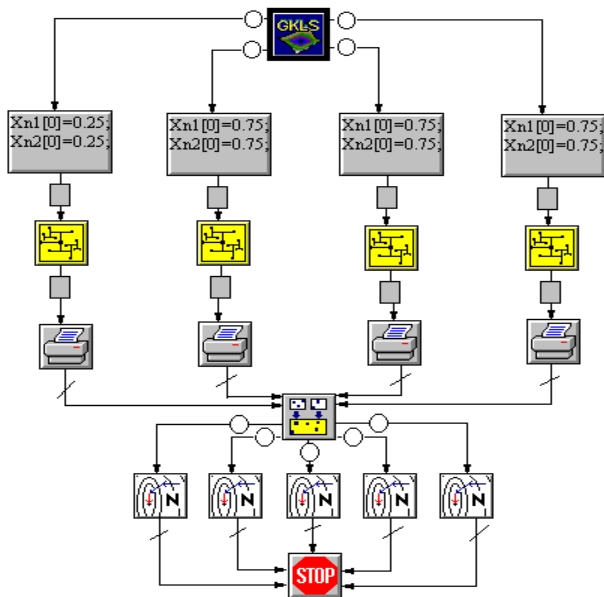


Fig. 2. Graph of a parallel program for global optimization

The programming environment of GSP-technology comprises the visual editor for drawing of graphs and defining data and modules, the graph compiler for generating C-source files from graphs and the C-compiler for generating of executable file. Execution environment of GSP-technology

uses Message Passing Interface (MPI) for parallel programs execution. Programs generated with GSP-technology can work on clusters and other systems with MPI support.

Each parallel branch is presented with dedicated MPI process.

To emulate shared memory model in MPI environment, a special memory manager is developed. It allocates memory for data dictionary, initializes program's variables, transmits data to and from the processes and frees unused memory. Memory manager is executed in dedicated MPI process. It is a program that receives data requests from different processes and reads/writes data to or from the memory. Memory manager eliminates memory conflicts between processes.

The parallel program can contain many processes. When there are hundreds or thousands of processes it is inconvenient or just impossible to draw such number of parallel branches on the graph. For such cases GSP-technology uses a special kind of graph nodes called "multitop".

Multitop is represented as one node on the graph and has three parameters associated with it: the module or graph being executed with many processes, the number of parallel processes (branches) represented by the multitop, and the name of the variable which holds the sequence number of each process generated by the multitop. The variable is used within the multitop's module or graph to define its actual function in the same manner as the process rank is used in MPI.

Fig. 3 shows an example of the graph which uses multitops to describe the program similar to that on the Fig. 2 running on 500 processes.

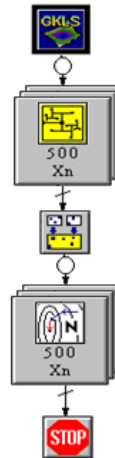


Fig. 3. Graph of a parallel program for global optimization with multitops

Large number of processes in parallel program is usually used to perform some similar tasks on different independent data without synchronization between the processes. Representation of such tasks as a multitop seems to be a tradeoff between the clarity and the compactness.

III. PRESENT STATE AND FUTURE DEVELOPMENT

For a long time the graph editor in GSP-technology was a desktop application. It comprised graph compiler as a component and was dependent on external C-compiler and database management system (DBMS). This had led to the difficulties in deployment of the system. To install the system in some new location (for example in laboratory classes) one should install the graph editor, then install and properly configure an external C-compiler and DBMS. Using a cluster as a target system for the programs built in GSP-technology requested the direct access to the cluster through the SSH protocol.

To make the use of the GSP-technology easier the web-version of the graph editor was developed. The web-server and DBMS were installed together on the same host and provided remote access to the editor. The editor worked with the database locally and had an SSH connection to the cluster. The main disadvantage of such a system is that the web-interface applies some restrictions to the editor making it less convenient for the users than a desktop application.

Cloud computing has made it possible to combine the rich interface capabilities of desktop graph editor with the centralized management of the whole system for many users. We are working on the development of the Platform as a Service (PaaS) system which will provide visual parallel programming with GSP-technology. PaaS system comprises one virtual machine which hosts the web-server and database and has an SSH connection to the cluster. Many virtual machines can also be run in the same cloud environment each hosting the desktop version of the graph editor. As the database is the same for the web-based and desktop graph editors, it is possible to work on the same project for the team of developers using both versions of editors concurrently.

Some additions have been made to the desktop version of the system. The registration and subsequent authorization of the users running the desktop version was added. During the logon process the user can see the status of other users (online/offline or working with the same project as the current user). All changes made by the user during the session are logged to the database. It is necessary for producing the snapshots - the states of the project development process when some valuable results are achieved, for example, for saving the intermediate working versions of the algorithm which is under development. Another goal of user activity logging is to track the changes made by different users and by the same user in different versions of the system. With logging it is much easier to remember what exactly you have changed while working with the project from the other place (for example, from home) or to understand (and also to explain) the changes made to the graphic model of the program by some other person.

Visual programming can benefit from cloud computing as it provides the capability of shared development that text programming lacks. With text programming the basic tool of team software development is version control system. The concurrent editing of the same file with source code is practically useless. The basic approach is the division of project to smaller tasks, assigning them to different developers and combining results with version control system. With visual

programming tool running in the cloud it becomes possible to work on the same graph concurrently. Such shared work is meaningful and can be convenient due to the compactness of visual representation. Editing the same graph concurrently you can easier develop the proper solution of a problem or find the error in a program faster. The visual editing process is similar to the process of discussing something, while graphically illustrating the main ideas being discussed. The visual programming in such implementation gains the features of the visual modeling.

The main issues to resolve in PaaS visual programming service being developed are the following: concurrent work of several users with one project, versioning, compiling and running parallel programs from the desktop virtual machines on the cluster, optimization of the communication between the system and the cluster.

There are also many tasks in the development of the GSP-technology: dynamic processes creation in MPI programs generated by GSP-technology, direct local data exchange between the parallel branches, creation of graph compilers for other parallel programming technologies like OpenMP and CUDA, making interfaces with other programming languages, technologies and libraries in order to leverage code reuse.

REFERENCES

- [1] H. Goma, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison Wesley Object Technology Series, Reading MA, 2000.
- [2] N.I. Polikarpova, A.A. Shalyto "Automata-based programming," SPb.: Piter, 2009 [Поликарпова Н. И., Шалыто А. А. Автоматное программирование. СПб:Питер, 2008. – 167 с.]
- [3] A.N. Kovartsev, V.V. Zhidchenko, D.A. Popova-Kovartseva, P.V. Abolmasov "The basics of graph-symbolic programming technology," Proceedings of the Open semantic technologies for intelligent systems (OSTIS-2013) III international conference, pp. 195-204, 2013 [Коварцев, А.Н. "Принципы построения технологии графосимволического программирования" / А.Н. Коварцев, В.В. Жидченко, Д.А. Попова-Коварцева, П.В. Аболмасов // Труды II Международной научно-технической конференции «Открытые семантические технологии проектирования интеллектуальных систем». -2013. - С. 195-204.]

Procedures classification for optimizing strategy assignment

Olga A. Chetverina

ZAO MCST

Moscow, Russia

Chetverina_o@mcst.ru

Abstract— Optimizing compilers make significant contribution to the performance of modern computer systems. Among them VLIW architecture processors are the most compiler-dependent, since their performance is ensured by effective compile time scheduling of multiple commands in a single clock. This leads to an eventual complication of VLIW compilers. Taking as an example optimizing compiler developed for the Elbrus family processors, it runs consequently over 300 stages of code optimization in basic mode. Such an amount of stages is needed to obtain decent performance, but it also makes compilation quite time consuming. It turns out that the main reason for compilation time increase when using high level compilation is applying some aggressive unversable code transformations, which eventually leads to code size increase that is also unwanted. In addition, there remains the problem of using a number of optimizations that are useful for rare contexts. To reach the objectives, namely increasing performance, decreasing compilation time and code size, it is reasonable to choose an appropriate strategy on an early compilation stage according to some procedure specific characteristics. This paper discusses the procedures classification problems for this task and suggests several possible solutions.

(Abstract)

Keywords—*optimizing compiler, optimizing phases sequence, performance tuning, reducing compilation time, procedures classification*

I. INTRODUCTION

To obtain decent performance modern optimizing compilers apply a huge sequence of code transformations. Usually compilers use a fixed optimization sequence for all procedures according to optimization level (-O0, -O1, -O2, -O3) and each optimization stage tries to improve performance of available code segments using statistically proven heuristics which leads to suboptimal results in most cases [1, 2]. In order to achieve the best possible performance for a given program it is important to find the most suitable optimization sequence for each procedure. This could be done with iterative approaches, which compile procedures in a given program using different optimization sequences with either executing the resulting code [3,4] or estimating the execution time [5] and choosing the best one. Although both techniques achieve good performance results on a number of tasks, their weak spots is a need of a

large compilation time which is not always acceptable and a necessity to execute tasks on appropriate input data so that the training runs would match the further execution in terms of branch probabilities and code coverage. The importance and difficulty of constructing a good training input data can be demonstrated with profiling data that was collected using train execution of the spec2000 benchmark [6] using Elbrus compiler. It was found out for this benchmark that applying a low-optimizing sequence to the procedures with zero train profile data leads to a 6% performance degradation of CFP tasks of spec2000 on average. The biggest decelerations occurred on 179.art (-18%) and on 301.apsi (-47%), where the reason for 301.apsi degradation is that one of its main procedures never executes during train run. As for huge applications it is often too difficult to generate good train data, which will cover all important parts of code, moreover, for some types of code like libraries or operational system it is nearly impossible. Also it should be mentioned that in most cases high compilation time corresponds with the resulting code size growth, this happens because most time-consuming phases including hyper-blocks construction, scheduling and loop software pipelining are located in the end of optimization line and the time they work corresponds with the size of the intermediate code that was made as result of different aggressive loop and acyclic transformations such as splitting, peeling, tail duplication etc.

Earlier researches in the field of iterative compilers [7,8] offer techniques that allow to construct a set of optimization sequences that cover the given procedures space rather well. In those works to minimize the needed execution time authors choose a possibly small set of options or sequences that show performance increase on most tests. To reach good performance results with affordable compilation time and resulting size of code and avoid the need of training executions it is reasonable to try to choose a compilation sequence from such a set on an early compilation stage using some characteristics of the procedure. The main goal of this research is to explore and construct the possible methods of procedures classification that would allow to perform this objective.

First of all it would be shown that to make a good selection of optimization sequences for a set of procedures using characteristics a compilation quality functional is needed (section 2). It would also be explained how to construct a functional to take several factors into consideration, like execution time, compilation time, resulting code size and other

possible limits. Then the task of predicting good sequences selection for a given number of procedures would be formulated in terms of minimizing constructed quality functional (section 3). After a list of main existing methods of classification and clusterizations would be described and given a possible one that allows to solve the task. In section 4 some experimental results would be provided.

II. COMPILATION QUALITY FUNCTIONAL

To make a statistical solution of procedures types selection a large training set is needed. For this purpose all procedures of spec2000 benchmark with a full input data were used. The reason for this pack choice is that it is well balanced in terms of different types of tasks and is used as a performance benchmark for most high-performance computers. The steps for solution is to choose the best sequences assignment for the training set using full statistic on compilation, execution or other important characteristics and then to make an attempt to predict it using only procedures information available on early compilation stage.

Any type of classification and clusterization methods perform allocation of areas in parameters space, which are then respectively called classes or clusters and could be used to make some assignment of type, in our case an assignment of optimization sequence. Using an example from Table I it could be easily seen that a need to construct a quality functional comes up even when the only goal of classification is to minimize execution time.

TABLE I. EXAMPLE OF SEQUENCE CHOICE

	<i>Sequence 1 time</i>	<i>Sequence 2 time</i>	<i>Best sequence</i>
Procedure 1	100	50	2
Procedure 2	95	100	1
Procedure 3	100	105	1
Sum time	295	255	2

Suppose there are 3 procedures that hit the same area in parameters space, in the shown example the best sequence choice for 2 out of 3 procedures would lead to decrease of performance both in sum and on average. It could be assumed that procedures with different optimal sequences should be in different areas but actually this assumption is wrong because even the same procedure with different input data could lead to different best choices results. This means that there is a need to construct a numerical evaluation method that would qualify the sequences assignments on the whole set of procedures. The most common technique to formalize the understanding of the best choice is to construct a functional, which reaches minimum at decision point. In this case the domain for such functional is an *assignment space for procedures*:

$P = \{p_1, \dots, p_n\}$ – all procedures in a set

$L = \{l_1, \dots, l_k\}$ – the list of optimization sequences,

$F(l(p_1), \dots, l(p_n)) \rightarrow R$ – a functional defined on the space L^n , where $l: P \rightarrow L$

To minimize the execution time the following functionals could be chosen:

$exe(p_i, l(p_i))$ - execution time of procedure p_i when compiled using $l(p_i)$ sequence, then

$$F(l(p_1), \dots, l(p_n)) = \sum_i exe(p_i, l(p_i)) \quad (1)$$

$$F(l(p_1), \dots, l(p_n)) = \prod_i exe(p_i, l(p_i)) \quad (2)$$

A functional that considers not only the execution time, but also compilation time could be constructed:

$comp(p_i, l(p_i))$ - compilation time of procedure p_i when compiled using $l(p_i)$ sequence

$$F(l(p_1), \dots, l(p_n)) = (\sum_i exe(p_i, l(p_i)))^r (\sum_i comp(p_i, l(p_i))) \quad (3)$$

This functional describes the acceptable ratio of performance loss and compilation gain, larger values of “r” mean higher importance of performance over compilation. Though even with infinite value of r compilation could be reduced in case if 2 sequences produce the same code in terms of execution time. Other important limitation as code size could be introduced into quality functional similarly.

III. FUNCTIONAL MINIMIZING CLASSIFICATION

Suppose a quality functional was already chosen, then classification task could be formulated in the following terms:

$P = \{p_1, \dots, p_n\}$ – all procedures in a set

$L = \{l_1, \dots, l_k\}$ – the list of optimization sequences,

H – the space of procedures characteristics

$Ch: P \rightarrow H$ – assignment of characteristic vector for procedures

$F(l(p_1), \dots, l(p_n)) \rightarrow R$ is defined on the space L^n , where

$l: P \rightarrow L$

Then the classification is an allocation of areas S in the space H with a sequence vector in L that produces a constant assignment for each area S , that is:

$$\forall S \quad l(Ch^{-1}(S)) = const$$

The goal is to make a classification (with some minimal number of training elements in the area = q), that minimizes the given functional:

$$F(l(p_1), \dots, l(p_n)) \rightarrow min \quad (4)$$

To substantiate the statistical approach it is reasonable to require for each procedure p_k having a locality D in characteristic space containing at least q points for which

$$H(p) = \begin{cases} l(p_k), p \in D \\ default, p \notin D \end{cases} \quad (5)$$

$$F(H(p_1), \dots, H(p_n)) \leq F(default, \dots, default)$$

A. Procedures characteristics

As was mentioned earlier the major use of such early compilation stage sequence prediction is expected on codes that for some cases are not suitable for training execution. So the goal is to choose a number of characteristics that work well enough to predict a good optimization sequence and do not depend on precise profile information. To choose the best set different characteristics were considered and using correlation matrix the most valuable were picked and normalized. The best characteristics that were found to predict the optimal compilation sequence with no train profiling information are:

- number of operations in the procedure;
- average node size, which in some sense stand for the branch frequency;
- number of call operations;
- maximum loop level in a procedure;
- average operation counter, which could also be considered as *procedure density*;
- percentage of operation of field reads;
- percentage of operations with floating point;
- percentage of operations that calculate an address for a read.

Most of those are profiling data independent, though the average operation counter is not. In case of no train profile information Elbrus compiler uses a predicted profiling based on statistical information. It was found to be good enough to use this static profiling for classification.

B. Ideal theoretical solution

First of all for the given training space that includes all characteristics, which are used in quality functional, an optimal solution that stands for the minimum functional point could be calculated. For the chosen functional (3) and the considered lines finding the minimum required making about $2 \cdot n$ steps of gradient descent, that is $2 \cdot n$ steps, where on each we make a change of a coordinate in assignment vector that gives the maximum functional value decrease. To check the stability of the resulting vector in L^n several starting points with the constant assignment of each line for all set of procedures were used. The solution is a vector with n coordinates where n is the number of procedures in the training set:

$$(l_{b_1}, l_{b_2} \dots l_{b_n}) \quad (6)$$

Sequence vector (6) would be called the *optimal theoretical vector of sequences* for procedures P , where l_{b_i} is the *optimal theoretical sequence* for procedure p_i . It should be noted that

l_{b_i} would not always afford the best performance or performance with compile time result on procedure p_i . It is optimal only in sense of the whole set of considered procedures, which is due to functional minimum.

As it would be shown in experimental section solution (6) doesn't always lead to best results on a real run when assigning the corresponding compilation sequences for all procedures in program, and therefore it is declared theoretical. This occurs because statistical information for each procedure is collected with simultaneous sequences assignment for other procedures in the program, modification of those procedures sometimes leads to other memory usage interaction and as a result to different execution time. The only way to completely avoid this effect is to collect statistical information for all possible configurations, which is not feasible and even to be partially used requires availability of information for all additionally executable procedures to make the right choice for the given one. Therefore, it was decided to drop out this fact in the currently constructed solution, though keep it in consideration for future researches in case of -fwhole-program compilation mode.

C. Existing classification and clusterization methods

Unlike to methods of clusterization [9] in this situation it is impossible to construct a metric that would determine the valuable in terms of our needs distance between procedures. The reason is that the distance between couples of procedures would depend on the other procedures in same cluster. For this case the clusterization methods allow to select areas according to only characteristic metrics, but it is possible only with appropriate characteristics normalization. The uniform normalization by itself works out bad for this task, though probably some techniques that use functional value movement with characteristic change could be developed.

Classification methods (support vector machine - SVM, Bayesian network) don't require to construct a metric that would divide classes. But as was mentioned before it is not enough to increase the possibility of picking the best sequence when using procedures characteristics for prediction. Though in the first attempt to make a classification solution a Bayesian network [10] has been tried. Although it showed a high percent of an optimal sequence prediction (above 95%) the resulting execution time of training tasks set increased by 21% on average. It was found out that the most frequently optimal sequence reduced the performance of some weighty procedures, which required a number of aggressive transformations to achieve acceptable performance. Due to this reason even a small percent of mistakes led to unacceptable result. Other considered methods have the same problem - the maximum that they allow is to add a weight to the mistake when choosing the wrong solution, which in our case means not optimal, but they don't differ the value of a mistake.

D. Procedures classification

To solve this problem a cluster error minimization algorithm was developed. First we construct the full error table. For each sequence l_{i_k} and for each procedure p_k the minimization error is the following

$$err[p_k, l_{i_k}] = \log(F(l_{b_1} \dots l_{i_k} \dots l_{b_n}) / F(l_{b_1}, l_{b_2} \dots l_{b_n})) (7)$$

For optimal sequence of procedure p_k functional

$$F(l_{b_1} \dots l_{i_k} \dots l_{b_n}) = F(l_{b_1}, l_{b_2} \dots l_{b_n}),$$

so error (7) is zero for the optimal sequence and could be zero or positive for the other sequences.

The main idea is to allocate on each step an area with new sequence assignment that would give a good functional value decrease comparing to the current. Which in terms of calculated errors would mean minimizing the summary error.

The clusters construction:

- Start.
- Assign the default sequence for each procedure. Calculate sum error W for all procedures.
- Repeat:
 - Choose not marked procedure p with maximum current error and the optimal sequence l_{p_k} .
 - Calculate the distances to all characteristics borders.
 - Calculate sum error for all space with l_{p_k} .
 - Define it as a current cluster.
 - Repeat for each characteristic:
 - Repeat until cluster size $\geq q$ and the calculated error decrease: with coefficient $t_1 < 1.0$ decrease the distance to one of the borders of the cluster
 - Repeat until the calculated error decrease: with coefficient $t_2 < 1.0$ increase the distance to one of the borders of the cluster
 - Accept the cluster if it decreases error by $dW \geq t_3 * W$. Mark the starting procedure with the flag.
- End.

The constructed areas are q – *dimensional* rectangles and could intersect. To choose the sequence for a procedure with the set of constructed cluster borders we take the sequence that corresponds with the last cluster that procedure belongs to. Parameters t_1, t_2, t_3 are heuristically chosen so borders movement would capture enough procedures to get more precise direction of error change.

Classes' construction can be started with any sequence; in proposed algorithm the default sequence was chosen because it is optimal on average. Also was made an attempt to start cluster construction with all procedures and choose the one that gives the highest minimization of functional value.

The received clusters with both attempts are very similar, though the last one is much more time-consuming. The other variant that was tested is the binary search of boundaries. This gave also a close result, and this mechanism could be assumed preferable because of no border parameters need.

The possible weakness of proposed classification is the absence of functional monotony by parameter coordinates; this could lead to inaccurate border calculation. Parameters t_1, t_2 or binary search of boundaries should reduce this effect because in both cases first steps in parameter space are big in terms of considered procedures number thus are statistically proven. One more limitation of constructed classes is that they are q – *dimensional* rectangles, though with the allowed intersection could actually take other forms. This could perform less accurate area selection but further significantly reduces required time for compiler to compute the proper class for a procedure.

IV. EXPERIMENTAL RESULTS

The proposed clusterization was implemented in Elbrus compiler. As the training set 9183 procedures of spec2000 benchmark were used. The whole amount of procedures in the given pack is much greater but it was possible to use only the procedures with a measurable execution time. In all cases the clusterization was constructed using full information on execution and compilation time corresponding with each sequence assignment to each procedure, then the solver, that computes procedures characteristics on early compilation stage and chooses the cluster according to calculated borders, was developed in the compiler. The assignment takes place in the end of interprocedural compilation stage, thus the time required for the sequences selection is included in whole task compilation time and is counted in the received compilation speedup.

As was already explained, the effectiveness of sequences assignment depends not on the highest probability of choosing the best line for procedure alone but on integral characteristic for the whole set. So to show the quality of constructed clusterization it is reasonable to consider all the tasks and not procedures separately. For this purpose results of implementing sequences assigned by optimal and clusterization selections were compared on whole spec2000 benchmark tasks. In this case we used functional that minimizes only performance time

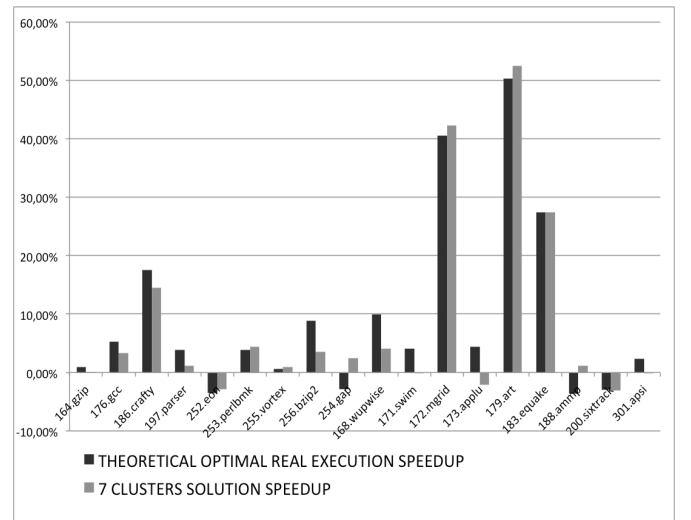


Figure 1. Optimal and cluster solution, spec2000, 7 clusters

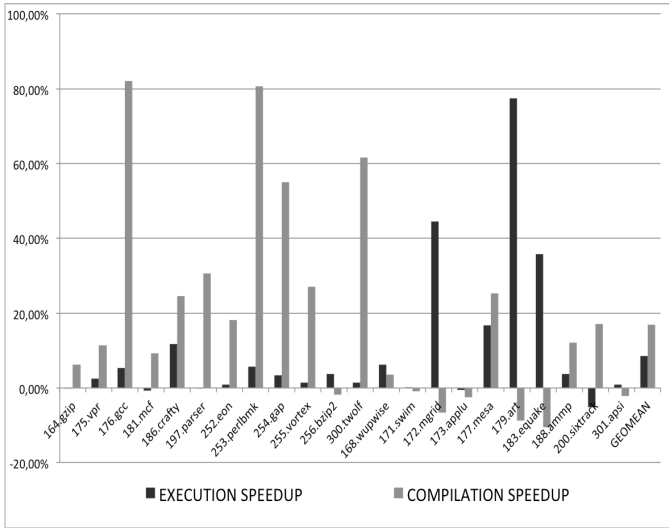


Figure 2. spec2000 no train execution, 5 clusters

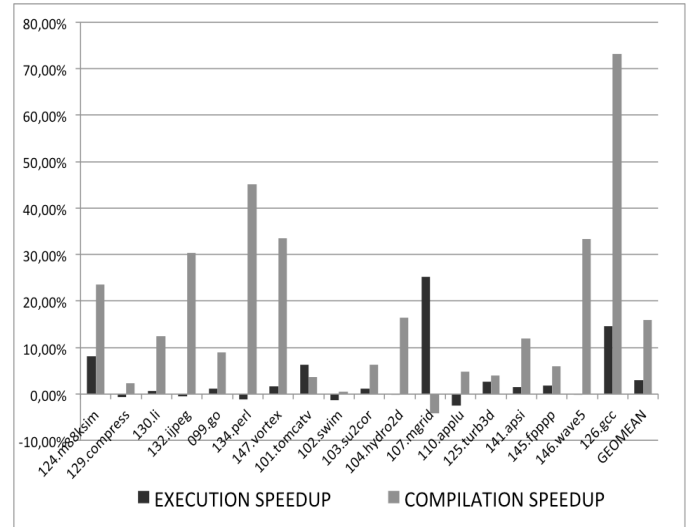


Figure 3. spec95 no train execution, out of train set, 5 clusters

(1) and constructed 7 clusters. The result is shown on Figure 1. As it was already discussed in section III “Ideal theoretical solution” the optimal solution for the tasks was combined of optimal theoretical sequence for each procedure. It was noted that because of the memory interaction some tasks, for example, 200.sixtrack, slowed down even with applying this optimal solution. As the result the real measure of optimal solution gained almost 5% less performance increase than it was supposed to be according to theoretical calculations. The same comparison with functional (3) – considering both execution and compilation time yielded worse clusterization results, it occurred mainly because a large amount of procedures are not executed and optimal solution gave much better compilation time results on them.

When using functional (3) most effect was achieved after constructing first 5 clusters. The corresponding sequence assignment for those clusters reduced compilation time by 17% on average and increased performance by 8.5% on the training set. Figure 2 shows the improvement obtained on certain tasks of spec2000 benchmark. As a test pack for the clusterization spec95 [6] benchmark was used. The execution and compilation result for this pack is shown on Figure 3. The average increase of performance reached 3% and the average compilation time decrease was over 16%.

Measured results prove effectiveness of classification algorithm, though due to the absence of functional coordinate monotony it is not proved that the best possible solution is received. Another question is the quality of available procedures characteristics choice, which showed to be good enough for the considered set of compilation sequences but could appear not to be representative to make quality selection from different set of sequences.

V. FUTURE WORKS

Results presented in experimental section show the possibility of good sequence prediction using classification methods. But some questions should be cleared and researches

to be done. First, it could be possible to make hierarchical clustering if inserting some metric that would allow to avoid problems with sporadic points that give inaccurate values for some reasons, this could allow better cluster borders calculation. Another question is how to construct the best training set in sense of avoiding procedures execution interaction. As it can be seen on Figure 1 the execution profiling of the whole task with one sequence can lead to errors in future procedure sequence selection. Also it could be more effective to combine sequences construction with some estimation of future prediction possibility using available procedures characteristics. Finally, there could be done some researches on ascertainment if the found procedure characteristics are good enough to provide maximum possible potential in best classes allocation.

VI. CONCLUSION

This paper introduces problems that come up on the way to develop automatic optimizing sequence selector that provides performance increase and reduces the needed compilation time for each procedure. Necessity of a quality functional on the space of all possible assignment is explained. Also it should be mentioned that such functional could include any possible limitations besides compilation and execution, in some cases it could be valuable to limit code size increasing or reduce the number of registers that are allowed for code planning. The last limit could be useful to lower register spill fill blocking between the calls and returns from large procedures.

An effective algorithm that can be used to select clusters in the procedures characteristics space is suggested.

The classification methods were implemented in Elbrus compiler. It was shown that a good optimization sequence could be chosen even when it is impossible to execute the code and no train profiling information is available. The results were achieved and introduced using spec2000 and spec95 benchmarks.

REFERENCES

- [1] Kulkarni., W.Zhao, H.Moon, et al. "Finding Effective Optimization Phase Sequence", [A]. Proc. of ACM SIGPLAN 2003 Conference on Languages, Compilers and Tools for Embedded Systems, US:2003.
- [2] Spyridon Triantafyllis, Manish Vachharajani, Neil Vachharajani, David I. August. "Compiler optimization-space exploration. Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization", March 23-26, 2003, San Francisco, California.
- [3] Keith D. Cooper, Alexander Grosul, Timothy J. Harvey, Steven Reeves, Devika Subramanian, Linda Torczon, Todd Waterman. "ACME: adaptive compilation made efficient", LCTES '05 Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, Pages 69 – 77
- [4] Prasad A. Kulkarni , David B. Whalley , Gary S. Tyson, "Evaluating Heuristic Optimization Phase Order Search Algorithms", Proceedings of the International Symposium on Code Generation and Optimization, p.157-169, March 11-14, 2007
- [5] Prasad A. Kulkarni, Michael R. Jantz, David B. Whalley, "Improving both the performance benefits and speed of optimization phase sequence searches", LCTES '10 Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems , April 2010
- [6] Standard Performance Evaluation Corporation, <http://www.spec.org/>
- [7] Suresh Purini, Lakshya Jain. "Finding good optimization sequences covering program space". Transactions on Architecture and Code Optimization (TACO), January 2013.
- [8] M. Haneda , P. M. W. Knijnenburg , H. A. G. Wijshoff, "Generating new general compiler optimization settings", Proceedings of the 19th annual international conference on Supercomputing, June 20-22, 2005, Cambridge, Massachusetts
- [9] Jain, Murty and Flynn: "Data Clustering: A Review", ACM Comp. Surv., 1999.
- [10] Judea Pearl, Stuart Russell. "Bayesian Networks". UCLA Cognitive Systems Laboratory, Technical Report (R-277), November 2000.

Towards Finding Several Bugs at Once by CEGAR

Vitaly Mordan

Institute for System Programming
Russian Academy of Sciences
Moscow, Russia
mordan@ispras.ru

Vadim Mutilin

Institute for System Programming
Russian Academy of Sciences
Moscow, Russia
mutilin@ispras.ru

Abstract — Every program should meet a lot of requirements. In order to prove programs correctness, static verifiers, which are based on Counterexample Guided Abstraction Refinement (CEGAR), check programs against each requirement separately, since in general case checking of one requirement may interfere with another. In that case a lot of resources is wasted, since similar actions are performed during checking of different requirements, such as construction of abstraction of the program in CEGAR. The paper suggests preliminary ideas of CEGAR approach modifications, which are aimed at checking programs against several requirements at once, taking into account that every program may contain several bugs violating the same requirement. The suggested CEGAR modifications potentially can significantly reduce total verification time and get the same results as basic CEGAR.

Keywords — static verification, counterexample guided abstraction refinement, bug, requirement.

I. INTRODUCTION

Static verification is a formal means of checking program source code without its execution by exploring all possible program paths [1]. The main benefit of static verification is that it aims at proving correctness of the software instead of simply finding frequent bugs. The main disadvantage, which makes it much less applicable in practice, is a large amount of required resources (such as processor time and memory), especially for large software systems.

The other part of the problem is that every single program in a big software system may contain any number of different bugs. In order to find them with help of static verification, all specified requirements to a program must be checked. This obstacle further increases required resources for the verification process.

Let us consider an example, based on using of static verification in practice. Linux Driver Verification Tools (LDV Tools) are an open source toolset for checking correctness of Linux kernel modules against rule specifications (requirements) with help of different static verifiers [2]. It has already helped to find more than 190 bugs in Linux kernel modules [3]. The process of verification of all Linux kernel modules with help of LDV Tools against a single rule specification with static verifiers BLAST [4] or CPAchecker [5] takes about 2 days. But the number of those rule specifications is more than 50 and more are under development. That is why only the major releases of Linux

kernel are checked with help of LDV Tools and only against a few selected rule specifications.

This paper presents some ideas of new verification method, which potentially can significantly reduce required time even if the number of requirements to be checked is growing. Multi-Aspect Verification is aimed at checking more than one requirement (or aspect) at once. The main demand to this method is to prove correctness of the program against requirements and to find their violations as CEGAR approach, but faster. Also this method should consider that each specified requirement may be violated more than once. The suggested method will be implemented and the experiments will be conducted in order to evaluate it, corresponding results are going to be published later.

Next section describes basic static verification approach and definitions, which are used in the paper. Section III presents existing extensions of basic static verification approach. In section IV the main ideas of Multi-Aspect Verification are suggested.

II. BACKGROUND

One of the most scalable static verification approach is **Counterexample Guided Abstraction Refinement (CEGAR)**, which was demonstrated in Competition on Software Verification (SV-COMP) [6-7]. We took CEGAR as a basis for our static verification technique.

A. Aspect definition

Aspect is a formal representation of checked requirement to the program [9]. Aspect represents what we intend to check in the program. For example: “*allocated resources should be correctly freed*”. Aspects are also called safety properties or rule specifications (LDV Tools). Hereafter only notion of aspect will be used.

Error location is an internal representation of an aspect for static verifier. Error location represents what static verifier is checking during the analysis. For example, predefined function *error()* or predefined label *ERROR*, which correspond to an aspect violation.

Verification task is a task for a static verifier, built on a program and an aspect. In CEGAR verification tasks usually are represented as reachability tasks, which are aimed at proving, that corresponding error location cannot be reached from specific entry point in the program (for example, call of function *main*).

Error trace is a sequence of operations in program source files that leads from a specific entry point to the aspect violation (which is represented as reached error location). Error traces are also called static verifier traces [13].

Verdict is a result of solving verification task by a static verifier. Usually there are 3 possible verdicts:

- *Safe*: a program is correct against a specified aspect;
- *Unsafe*: an aspect is violated in a program (corresponding error trace was found);
- *Unknown*: static verifier cannot solve a given verification task.

Since static verification is an undecidable problem in general case, verdicts *Unknown* are unavoidable even if we consider that static verifiers are free of bugs. That is why static verifiers are working with limited resources in practice. Therefore verdicts *Unknown* can be divided into 3 types:

- 1) Time limit exhaustion.
- 2) Memory limit exhaustion.
- 3) Abnormal termination of static verifier. First, verification task can be incorrectly created (for example, program source code is not compiling). Second, static verifier can contain bugs. Third, static verifier cannot decide some task and terminates itself.

B. Counterexample Guided Abstraction Refinement

CEGAR approach is based on the following notions.

Control Flow Automaton (CFA) [4] is a graph representing a program, in which nodes correspond to program locations and edges correspond to program operators.

Since it is usually impossible to analyze a precise model of a program in reasonable time, an abstract model of the program is analyzed.

Concrete state [4] consists of variables assignment, which assign to each variable its concrete value, and current program location.

Abstract state [4] is a set of concrete states of a program.

Abstract Reachability Graph (ARG) [4] is an abstraction of a program, in which nodes correspond to abstract states and edges correspond to edges from CFA.

Verification fact [8] result of the verification process (possibly intermediate), that is necessary for verifying specified aspect.

Abstraction precision (precision) [8] verification fact, which instructs the analysis, which information should be tracked and which information should be omitted in abstraction of the program. For example, in predicate analysis precision defines, which predicates should be tracked, in explicit value analysis precision defines, which variables should be tracked. Thus precisions defines the current level of abstraction in ARG.

CEGAR algorithm is presented in Fig. 1 [1]. At the beginning ARG is built based on initial precision (for example, empty precision). If no specified error location was reached, then the program is safe. Otherwise the found counterexample is checked for feasibility. If it is feasible then an error trace is

built for found counterexample and analysis terminates. Otherwise the precision is refined based on infeasible counterexample and the CEGAR loop is continued.

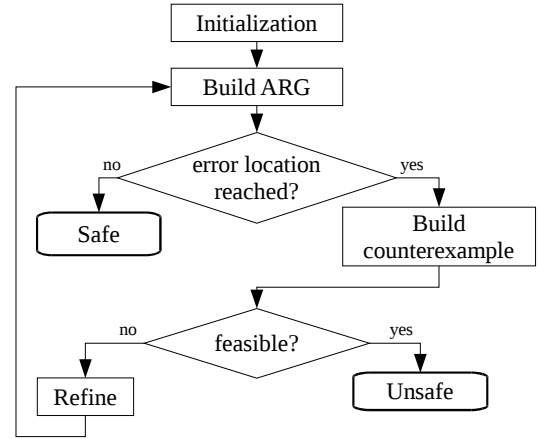


Fig. 1. CEGAR approach.

C. Example

Let us consider an example. There are three aspects:

- 1) Aspect_1: “all allocated resources by `usb_alloc_urb()` should be freed by `usb_free_urb(urb)`”.
- 2) Aspect_2: “same mutexes should not be acquired or released twice in the same process”.
- 3) Aspect_3: “offset should not be greater than size of the array”.

Verification task is created based on program source code and specified aspect. There selected aspect corresponds to some error location. For example, aspect_i can correspond to error label `ERROR_i`, $i=1,2,3$. Then aspect_3 may be represented by auxiliary checks in verification task:

```

if (offset > size)
    ERROR_3 : error();
  
```

In order to get verdicts for all specified aspects with basic CEGAR algorithm we need to prepare 3 different verification tasks and run CEGAR algorithm 3 times. Let us assume, that for some program corresponding to these aspects verification tasks get the following verdicts in CEGAR:

- Aspect_1: *Unknown* (time limit exhaustion);
- Aspect_2: *Safe*;
- Aspect_3: *Unsafe*.

In this paper suggested method should prepare only one verification task and find the same verdicts as CEGAR.

III. RELATED WORK

The idea of modifying CEGAR algorithm in order to reduce verification time is not new. By adding auxiliary actions in different points of CEGAR algorithm (Fig. 1) it is possible to solve specific tasks faster.

A. Regression verification

Even if a program was verified and absolutely correct, sooner or later it will be modified. For example, average number of changes per hour in Linux kernel modules is more

than 4 [10]. Every single modification of the program potentially may add new bugs. Such bugs are called regressions.

Regression verification is aimed at verifying program revisions in order to find regressions. One of the approaches to do it is to reuse verification results [11]. The main idea of these approaches is the following. CEGAR algorithm spends a lot of time to find needed level of program abstraction, which is defined by verification facts (for example, by precisions [8]). Those verification facts can be stored as result of the verification and then used on *Initialization* step of CEGAR algorithm (Fig. 1) to start analysis with known level of abstraction, for example, to verify next revision of that program. Experiments confirms [8, 11], that reuse of verification facts reduces time for regression verification.

At the same time in some cases reuse of verification results may increase time of the analysis. For example, abstraction may become too accurate for the new revision and analysis will take much more time than without reuse (for some verification task time with precision reuse was increased almost in 2 times [8]).

B. Conditional model checking

Verdict *Unknown* means that static verifier fails to solve the verification task, it is still unclear if the given program correct or not, resources were spent for nothing.

In order to solve this problem Conditional model checking approach is suggested [12]. Static verifier saves its result even if it cannot solve verification task for the whole program. This result is somehow should show, which parts of the program (for example, abstract states in ARG) were successfully verified and which were not. Then another static verifier (or the same with different configuration) takes this result at *Initialization* step of CEGAR algorithm (Fig. 1) and verifies only those parts of the program, which were not verified. Conditional model checking helps to solve problems, which cannot be solved by a single launch of static verifier [12].

C. Multiple error analysis

Any program can contain any number of bugs against given aspect. Basic CEGAR algorithm stops after it finds a first bug. In practice it significantly increases time for finding and fixing similar bugs.

In order to solve this problem Multiple Error Analysis (MEA) was suggested [13] (Fig. 2). Its main idea is to continue analysis after finding error trace. Obviously, time for the analysis will be increased (in comparison with basic CEGAR), but time for finding and fixing similar bugs will be reduced.

MEA adds new type of verdict – *Unsafe-incomplete*, which means that some number of error traces were found, but the analysis was not completed (i.e. there may be more error traces).

The main issue of MEA is that it finds a lot of similar error traces [13]. For example, instead of 23 error traces representing different bugs 1998 were found. In order to solve this problem static verifier trace comparison algorithms were suggested [13]. The main idea of these algorithms is to compare only the parts of error traces instead of the whole error traces. Lately this idea

was extended for multi-level filtering. Three levels of filtering were suggested to reduce number of error traces:

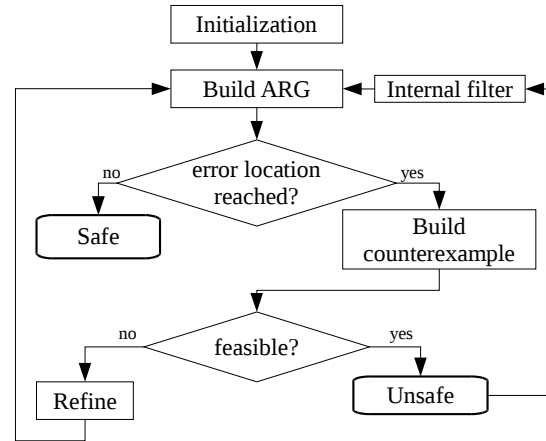


Fig. 2. MEA method.

1) **Internal filter.** Error traces are filtered in MEA algorithm itself (Fig. 2) based on their internal representation. This filter must be efficient and thus simple. For example, filter by ABE [14] in CPAchecker static verifier.

2) **External filter.** Error traces are filtered after MEA algorithm, taking into account specifics of programs under analysis and checked aspects. This filter may be more complex and thus less efficient. For example, filter by model functions in LDV Tools (see model functions comparison algorithm in [13]).

3) **Expert filter.** An expert marks up some bounds (i.e. operations in error trace) of the bug in a given error trace, then all error traces, that contain marked part (in terms of selected by expert comparison algorithm), are called equivalent to the first one and can be excluded from the further analyses (see semi-automated approach in [13]).

With help of three levels of filtering it is possible to minimize the number of error traces up to the ideal result (i.e. when every error trace correspond to different bug) with minimal effort of the expert.

IV. MULTI-ASPECT VERIFICATION

Multi-Aspect Verification (MAV) is aimed at configurable checking of several aspects for a given verification task at once. In this paper MAV is suggested as an extension of CEGAR algorithm, but potentially the same ideas can be applied to the other static verification approaches.

MAV solves the same verification tasks as CEGAR (i.e. reachability tasks). It is supposed that verification task has already been built for selected aspects. The main requirement to MAV is to get the same result for each aspect (i.e. prove correctness or find error traces), as basic CEGAR, but faster.

First problem, that must be resolved, is that in basic CEGAR error location is always one and it corresponds to one aspect. Obviously it is possible to check for several error locations (for example, check for error labels ERROR_1, ERROR_2 and ERROR_3 at once), but those error locations will not have any connection to checked aspects. So, if we are checking for so called combined error location (which is

consisted of usual error locations), we need to know which aspect is also being checked (to get verdicts as CEGAR).

Second, such combined error location makes it hard to determine, which corresponding error location is checked at the moment of time. If an error trace was found, it is unclear, what was violated (which aspect should get verdict *Unsafe*). Also it is impossible to limit resources for some aspect separately. Thus even if a single aspect cannot be checked by CEGAR (for example, it reaches time limit) then analysis for all aspects will be terminated.

MEA method cannot be used directly to solve MAV tasks. First, in order to get the same results as CEGAR, MAV should be able to find only the first error trace for each aspect. Second, MEA may get *Unsafe-incomplete* verdict meaning that violation of some aspect is missed as far as not all error traces were found. At the same time it could be useful to be able to find all violations for selected aspects at once.

So, MAV must satisfy the following requirements:

- to differentiate one aspect from another (in terms of error locations);
- to continue analysis after finding an error trace (verdict *Unsafe*);
- to find verdicts *Unknown* for each aspect separately and to continue analysis after them;
- to have an option for finding multiple errors for each aspects like MEA;
- to get the same verdicts for aspects as basic CEGAR (ideally);
- to consume less resources (in comparison with basic CEGAR).

A. Multi-Aspect Verification method

MAV algorithm was designed to meet the requirements (Fig. 3). It extends basic CEGAR algorithm and checks for combined error location.

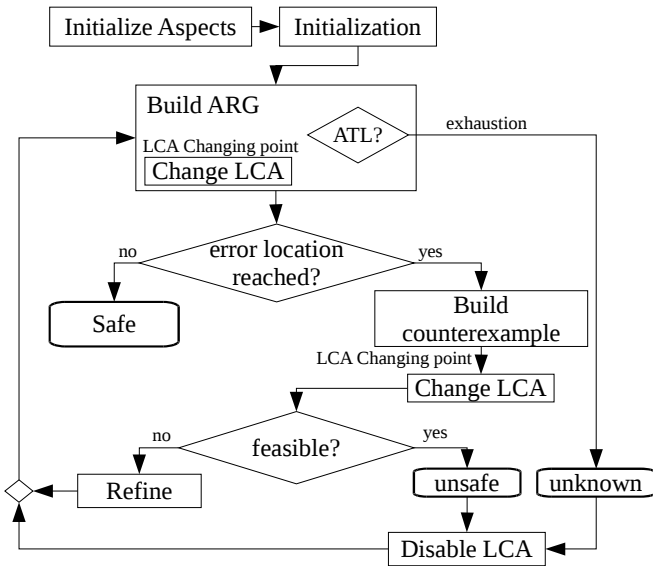


Fig. 3. MAV method.

Aspect (in terms of MAV) is the representation of checked aspect (which was defined in section II) into error location. Each aspect in terms of MAV have the following attributes:

- unique identifier (from the original aspect);
- corresponding error location (for example, specific label);
- aspect verdict;
- consumed resources (processor time, wall time, etc.);
- corresponding verification facts.

Hereafter the notions aspect from section II and aspect in terms of MAV represent the same entity (the only difference is that aspect in terms of MAV is an internal representation of aspect from section II for MAV algorithm).

At *Initialize Aspects* step (Fig. 3) links between aspects and error locations are created, attributes for each aspect are set to initial values.

Aspect verdict is an internal verdict of an aspect. Aspect verdict takes the following values:

- *checking*: aspect is currently being checked;
- *safe*: correctness of this aspect have been proved;
- *unsafe*: violation of this aspect has been found (along with corresponding error trace);
- *unknown*: algorithm suggests this aspect cannot be proved in the given verification task;
- *unsafe-incomplete*: violation of this aspect has been found, but the full analysis has not been completed (for MEA).

In order to know, which aspect is being checked (to change aspect verdicts, to keep records of consumed resources, etc.), the notion of **Latest Checked Aspect (LCA)** is suggested. **LCA** is the latest aspect, which was checked during the analysis.

During the ARG construction more than one aspect can be checked at once (i.e. ARG can be built based on few aspects at once). The following approximation is suggested: only one aspect is being checked at the moment of time and the latest checked aspect is LCA.

At the start of the algorithm LCA is unset, since no aspect was checked before. Then all time of the analysis can be divided by **LCA changing points**, which are moments of time, when LCA changes. Current checked aspect is considered equal to LCA until new LCA changing point.

LCA changing points can be added after *Build counterexample* step (it is always possible to determine which aspect was violated based on counterexample) or at *Build ARG* step (for example, in explicit value analysis it is possible to bind LCA changing points to specified variables changes). The main idea of these points is that they represent that algorithm is building ARG for the selected aspect. After that it is possible to divide analysis time line for **LCA intervals** which are time intervals between two LCA changing points. LCA intervals are used for calculations of time, consumed by each aspect. At *Change LCA* step (Fig. 3) time of the interval is added to new LCA time.

In general case verification facts for each aspect are obtained after *Change LCA* step for new LCA. For example, if we take precisions as verification facts they can be obtained for

and abnormal termination of static verifiers are still remain as unresolved problems. In such cases for all aspects with verdict *checking* we get aspect verdict *unknown*. For example, if only one aspect exceeds memory limit we could expect that MAV isolates it.

In order to achieve this MAV was extended based on ideas of Conditional model checking [12]. The main idea is the following. MAV saves intermediate result during its work. After termination of MAV the intermediate result is being analyzed. If analysis was not finished (for example, some aspect verdicts are *checking*) then algorithm will be restarted, but without aspects, that caused abnormal termination. Otherwise analysis is completed for a given verification task. This extension is called Conditional MAV (CMAV). CMAV method is presented in Fig. 5.

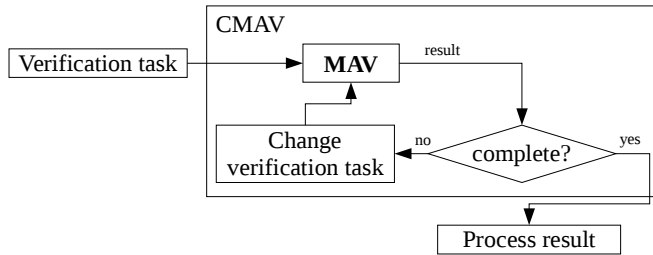


Fig. 5. CMAV method.

In order to save all actual information about verification process common format for intermediate results was suggested, in which every aspect keeps the following information:

- aspect identifier;
- current aspect verdict;
- consumed time;
- list of error trace identifiers, which were found for the aspect (only for aspect verdicts *unsafe* or *unsafe-incomplete*, in case of MAV with MEA after internal filtering);
- the reason of *unknown* (*unsafe-incomplete* in case of MAV with MEA). For example, reaching ATL.

Also LCA identifier should be stored.

The suggested information could be taken from aspect attributes and stored into the specified file. For each changes during the verification information in this file should be updated. Thus in case of any abnormal termination that file will contain all relevant information, including LCA, which might cause the termination.

Each launch of MAV algorithm is called an **iteration** of CMAV. After each iteration CMAV determines if analysis is completed based on the file with intermediate results. If algorithm was terminated abnormally, the reason of its termination should be determined and which aspect caused it. If there was global problem, then analysis should be completed, all aspects get aspect verdict *unknown*. If it was caused by some aspect, that aspect should get aspect verdict *unknown*. If analysis was not completed, then new iteration will be started, but it only should check aspects, which have previously had aspect verdict *checking*. In any case next iteration will get at least one less aspect to check. Thus, number of all iterations will be less or equal than the number of all aspects. After the last iteration is completed intermediate

information from all iterations should be united and presented as the final result for the given verification task.

Also the ideas of CMAV helps to solve problem, in which ARG is too complex. In that case after executing *Disable LCA* not all verification facts from previous LCA may be removed (or removing them requires too much time). Then it will be easier and faster to rebuild the whole ARG. For that reason Restart Time Limit (RTL) is suggested. RTL is ITL, which limits interval without LCA. In case of RTL exhaustion the whole algorithm will be restarted (new iteration of CMAV will start). In that case LCA is unset, because RTL limits intervals without LCA, but at least one aspect verdict is *unsafe*, *unknown* or *unsafe-incomplete*, since intervals without LCA are possible only after *Disable LCA* step. Therefore the next iteration of CMAV gets at least one aspect less to check.

Based on this consideration we think that CMAV satisfies the requirements in the beginning of this section.

V. CONCLUSION

The suggested method extends CEGAR approach and provides means to check several aspects at once. CMAV with MEA method can be used to find all potential violations of specified aspects in a program. Also CMAV method can be configured by using different options (specifying ITLs, removing different verification facts at disable LCA step, using option for MEA, choosing between LCA and SLCA) for specific verification task and user demands.

We plan to implement the suggested method as an extension of LDV Tools and CPAChecker static verifier and then to evaluate it on verification of industrial based software systems in future. Such experiments will show potential benefits as well as unexpected degradations in comparison with basic CEGAR in terms of verification time and verdicts, the method constraints will be revealed.

CMAV method was presented as CEGAR extension, but potentially the same ideas could be used for the other static verification approaches.

REFERENCES

- [1] M.U. Mandrykin, V.S. Mutilin, A.V. Khoroshilov. *Vvedenie v metod CEGAR — utochnenie abstraktsii po kontrprimeram* [Introduction in CEGAR — counterexample guided abstraction refinement]. Sbornik trudov ISP RAN [The Proceedings of ISP RAS], vol. 24, pp. 219-292, 2013 (in Russian).
- [2] Mutilin V.S., Novikov E.M., Strakh A.V., Khoroshilov A.V., Shved P.E. *Arkhitektura Linux Driver Verification* [Linux Driver Verification Architecture]. Trudy ISP RAN [The Proceedings of ISP RAS], vol. 20, pp. 163-187, 2011 (in Russian).
- [3] Bugs found in Linux kernel modules with help of LDV Tools. <http://linuxtesting.org/results/ldv>.
- [4] Beyer D., Henzinger T., Jhala R., Majumdar R. *The Software Model Checker Blast: Applications to Software Engineering*. Int. Journal on Software Tools for Technology Transfer (STTT), vol. 5, pp. 505-525, 2007. doi: 10.1007/s10009-007-0044-z
- [5] Beyer D., Keremoglu M.E. *CPAChecker: A Tool for Configurable Software Verification*. In Proc. Computer Aided Verification (CAV), LNCS, vol. 6806, pp. 184–190, 2011. 10.1007/978-3-642-22110-1_16.
- [6] Beyer D. *Competition on Software Verification*. In Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, vol. 7214, pp. 504-524, 2012. doi: 10.1007/978-3-642-28756-5_38

- [7] Beyer D. *Software Verification and Verifiable Witnesses*. In Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2015. 2015.
- [8] D. Beyer, S. Löwe, E. Novikov, A. Stahlbauer, P. Wendler. *Precision Reuse for Efficient Regression Verification*. In Proceedings of the 9th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE 2013), pp. 389-399, 2013.
- [9] Novikov E.M. *Razvitie metoda kontraktnykh spetsifikatsij dlya verifikatsii modulej yadra operatsionnoj sistemy Linux* [Development of a contract specification method for the verification of Linux kernel modules]. Dissertatsiya na soiskanie uchenoj stepeni k.f.-m.n. [PhD thesis], 2013 (in Russian).
- [10] Corbet J., Kroah-Hartman G., McPherson A. *Linux kernel development. How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*. <http://go.linuxfoundation.org/who-writes-linux-2012>, 2012.
- [11] D. Beyer, P. Wendler. *Reuse of verification results*. In Proceedings of the 20th International Workshop on Model Checking Software (SPIN 2013), LNCS, vol. 7976, pp. 1-17, 2013.
- [12] D. Beyer, T.A. Henzinger, M.E. Keremoglu, P. Wendler. *Conditional model checking: a technique to pass information between verifiers*. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE 2012), article 57, 2012.
- [13] V. Mordan, E. Novikov. *Minimizing the number of static verifier traces to reduce time for finding bugs in Linux kernel modules*. In Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCOSE 2014), editors A. Kamkin, A. Petrenko, A. Terekhov, Saint Petersburg, Russia, May 29-31. ISP RAS, Moscow, 2014. DOI: 10.15514/SYRCOSE-2014-8-5.
- [14] D. Beyer, M. Erkan Keremoglu, and P. Wendler. *Predicate Abstraction with Adjustable-Block Encoding*. In Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2010, Lugano, October 20-23), pages 189-197, 2010. FMCAD.

Acceleration of profile creation for three-dimensional vector video with GPGPU

Aleksandr Tsyganov

Chair of Computer technology

Samara State Technical University

244 Molodogvardeyskaya Str., Samara, 443100, Russian Federation

hitrolisk@gmail.com

Abstract—In the report the optimization of image similarity metric computation method for three dimensional vector video with general-purpose computations on graphical processor unit (GPGPU) is discussed. The use of stream processors in graphics accelerators and Compute Unified Device Architecture (CUDA) platform allows significant performance gain in comparison to calculations on general-purpose processors, while solving problems of computer vision and image similarity determination. The performance of the GPGPU metric value computation is measured and researched.

Keywords—three-dimensional video, graphical processor unit, computer vision, metrics, key points.

I. INTRODUCTION

Video playback systems for three-dimensional vector format need to determine parameter types of shader programs contained in the video stream. This can be accomplished by creating profiles for each video source type. Profiling is resource-intensive task and the calculations cannot be performed in real time while running the application for which the profile is compiled. The longest stage of the method is the metric calculation. The paper proposes to move its computation to graphics processing unit (GPU) in order to speed up the algorithm.

II. PROFILING METHOD

Method for automated profiling based on a comparison of images obtained with the original shader parameters and ones found after applying shaders with modified parameters.

For each shader it is necessary to find the correct types of values transmitted to its parameters. For implementation of stereoscopic effects, parameters containing projection matrixes are important. Thereby the problem is reduced to search such matrixes among parameters of the shader program. Parameters type search in the method is carried out by their search for each separate parameter. The assumption of correctness for the selected type is checked by similarity evaluation of images received from frame visualization of a video stream without modification of parameters and with modification of parameters according to the assumption made.

The selected frame V of the initial video stream is modified by transform $T(V, S)$ which changes set of shader parameters S concerning of which assumption was made about their certain type. The initial frame of V and the modified frame V' are

rasterized by $R(V)$ resulting in two images I and I' respectively. This images are represented by function of brightness in the given point $I = f_i(x, y)$. They are compared by using a metric. The result of applying this metric is the set D , consisting of two integral values, which are passed to the decision $A(D)$:

$$D = \{D_B, D_S\}, \quad (1)$$

$$A(D) = \begin{cases} 1, & D_B \leq b_m \cap D_S \geq s_m \\ 0, & D_B > b_m \cup D_S < s_m \end{cases}, \quad (2)$$

where b_m and s_m are the boundary values of the metric components.

Metrics computing algorithm for two images processes raw data in a few steps. Under the original data we will assume two images obtained with initial visualization parameters I_o and with modified visualization parameters I_m . Two color histograms $H(I_o)$ and $H(I_m)$ are calculated from the original image by dispersion method. Initial evaluation of the distance between the images performed by using Bhattacharya distance $D_B(H_o, H_m)$. Second component of metric is specified by comparing sets of control points in the original image. Sets of control points P_o and P_m , received from the image I_m and I_o , respectively, are used to calculate the distance $D_S(P_o, P_m)$. Speeded Up Robust Features (SURF) method is used for point detection, the implementation of which is also available for GPU [1, 2].

III. GPGPU IMPLEMENTATION

The architecture of modern graphics cards is designed for vector operations with the data in the form of multi-dimensional arrays. This allows to achieve high memory speed when using SIMD vector processors with independent L1 and L2 caches. In comparison to a general purpose processor, GPU has fewer steps and a smaller amount of the conveyors cache. Exchange of data between video memory and general purpose memory is implemented via the PCI-E x16 bus. The sample data in the cache transfers through a 256-bit bus. As a result, the efficiency of scientific algorithms on the GPU depends on the efficient use of memory and cache [3].

The main purpose of the GPU method implementation is to minimize the number of data exchanges between video memory and general-purpose memory. Communication between the CPU and graphics core negatively affect

performance. To reduce the data used by the various stages of the algorithm, it is loaded into video memory only once. The result is also available in video memory for the following stages. The essence of the developed method is the efficient use of the cache and loading video streaming GPU cores uniformly. Transfer of resources between the stages of the algorithm is carried out through the video memory, as shown in Fig. 1, which speeds up processing using the GPU.

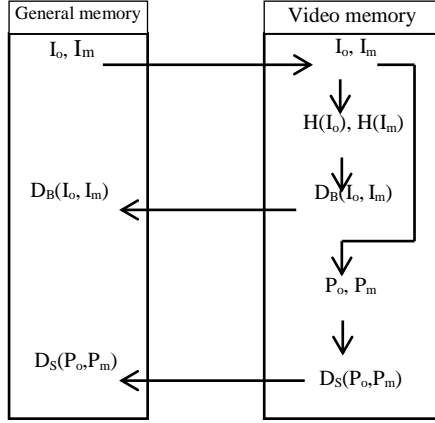


Fig. 1. Data exchange between general-purpose memory and video memory

Images I_m and I_o are loaded into video memory for processing. On their basis histograms are calculated to find the first components of the metric using Bhattacharya distance. The same source images used by SURF algorithm to calculate set of points, which are used as the basis for the second component of the metric. Only calculated components of the metric unloaded from video memory to general-purpose memory. Their size is extremely small, and video memory reading will not stop the process of computing on the GPU, resulting in high performance parallel computing.

IV. HISTOGRAM COMPUTATION

Calculation of the metric component D_B is performed by using the histograms $H(I_o)$ and $H(I_s)$ of corresponding images. The calculation of the histogram on the GPU can be performed using both classical shader programs, and using CUDA technology for general-purpose computation on the GPU. CUDA technology usage described in the works of Podlozhnyuk [4] and Shams [5]. These algorithms provide better performance than those based on the use of conventional means of graphical programming interfaces, as shown by Nugteren et al. [6]. Work of Fluck [7] is an example of the second approach.

Since the main objective is to accelerate the metrics calculation then most appropriate methods for histogram computation are based on CUDA. Such as method of Podlozhnyuk, that implemented in CUDA SDK. Method is cache effective and does not contain steps of data upload into shared memory that allows it to be integrated into the process of metric component calculation.

In this method, the original data is divided into blocks between threads executed on the GPU. Output data stream is stored in individual histogram. In the final pass all histogram are combined by different threads into one. To efficiently use

shared memory of streams each individual histogram is created in group of threads called rope. This allows to store histograms of a larger volume, up to 6 kilobytes on G80 hardware architecture.

Bhattacharya distance calculation based on the histogram for two sets of statistics. It is expressed by the following formula:

$$D_B = \sum_{i=0}^n \sqrt{H(I_o)_i H(I_m)_i}, \quad (3)$$

where n - the number of the histogram elements.

Calculation of histogram elements sums can be done by reduction of the initial data array on the GPU. It is proposed to use an optimized method of parallel reduction on CUDA, described by Mahardito et al. [8].

V. KEY POINTS DETECTION

The second component D_S of the metric calculated with SURF algorithm [9]. With its help search is performed for two sets of points P and P' , available in the original and the modified frames, respectively. The value of component determined by the following expression:

$$D_S = \frac{|P \cup P'|}{|P|}. \quad (4)$$

SURF is one of the most common and efficient image points search algorithms. It used in automatic object recognition and tracking, video recording, panoramic image combining and in many other areas of computer vision. The algorithm can process images in HD resolution at more than 30 frames per second.

SURF detects points by approximating the Hessian. Approximation performed by application of block filters to the image. It makes good use of the integral representation of the image I , which is determined by the following formula:

$$I(x, y) = \sum_{i=0, j=0}^{i \leq x, j \leq y} I(i, j). \quad (5)$$

The calculation of the integral image representation on the GPU is the longest stage of the SURF algorithm and can be implemented by the algorithm of the pyramid points as described in Terriberry et al. [10]

Construction of the integral image is the task of the prefix sum. Pyramid algorithm offers a solution to this problem on the GPU in two stages. At the first stage, pyramid images constructed extending upward, each of which divides into four parts half the width and height than the previous level. Image content is determined by three components of $U^{(k)}$, $H^{(k)}$, $V^{(k)}$:

$$\begin{aligned}
U^{(k)}(x, y) &= U^{(k-1)}(2x, 2y) \\
&+ U^{(k-1)}(2x + 1, 2y) \\
&+ U^{(k-1)}(2x, 2y + 1) \\
&+ U^{(k-1)}(2x + 1, 2y + 1),
\end{aligned} \quad (6)$$

$$\begin{aligned}
H^{(k)}(x, y) &= U^{(k-1)}(2x, 2y) \\
&+ U^{(k-1)}(2x + 1, 2y),
\end{aligned} \quad (7)$$

$$V^{(k)}(x, y) = U^{(k-1)}(2x, 2y) + U^{(k-1)}(2x, 2y + 1), \quad (8)$$

where k - level of the pyramid, x and y - coordinates of the image.

It requires two half-sum of $H(k)$ and $V(k)$ to calculate the sum of the even rows and columns, using formula:

$$X^{(k)}(x, y) = \sum_{i=0}^{x-1} H^{(k)}(i, y), \quad (9)$$

$$Y^{(k)}(x, y) = \sum_{j=0}^{y-1} V^{(k)}(x, j). \quad (10)$$

Using the obtained image pyramid, a reverse pass going from the top downwards. This value is used to calculate four different versions of the formula, that depend on the parity argument. For even x and y

$$W^{(k)}(x, y) = W^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right), \quad (11)$$

for odd x and even y

$$\begin{aligned}
W^{(k)}(x, y) &= W^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) \\
&+ Y^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right),
\end{aligned} \quad (12)$$

for even x and odd y

$$W^{(k)}(x, y) = W^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) + X^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right), \quad (13)$$

for odd x and y

$$\begin{aligned}
W^{(k)}(x, y) &= W^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) \\
&+ X^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) + Y^{k+1}\left(\left\lfloor \frac{x}{2} \right\rfloor, \left\lfloor \frac{y}{2} \right\rfloor\right) \\
&+ U^{(k-1)}(x-1, y-1).
\end{aligned} \quad (14)$$

The values of the top-level assumed to be zero.

Using the integral image, the key points are determined by searching the extremum of the Hessian determinant. Block filters used for this purpose as described by Bay et al. [9] Their GPU computation requires only 17 texture samples per pixel. Search for a local Hessian maximum can be made by the method of neighboring points $3 \times 3 \times 3$.

Each found key point is described by the descriptor, which is a normalized vector calculated using filters similar to the Haar block filter for Hessian. Sets of elements P and P' are compared using descriptors, which calculates the value of D_S with expression (4).

VI. PERFORMANCE EVALUATION

An experimental study with various sources of graphic information was carried to determine the performance gain of GPGPU implementation in comparison with the general-purpose processor implementation. Sources of graphical information were selected by statistics of streaming video services.

The first series of experiments aimed at assessing the dependence of the duration profiling on the recording. The results are shown in Fig. 2. As can be seen, the work time increases insignificantly, since longer records contains almost no new shader programs. However, there is a significant reduction in execution time by 8-12 times when using a GPU implementation.

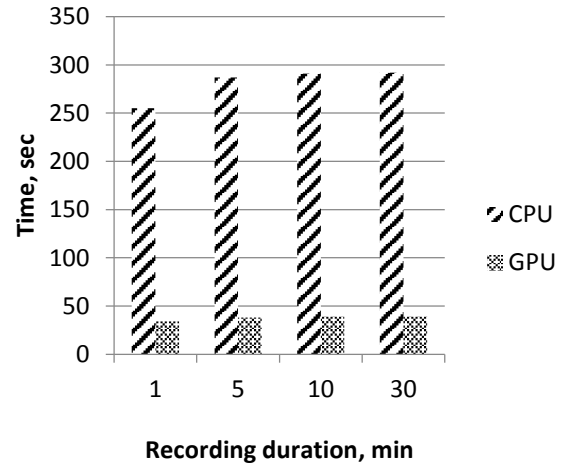


Fig. 2. The diagram of time depending on the duration of the record

Composition of the shader programs in each application is heterogeneous. The main feature affecting the complexity of the specific shader program analysis is the number of its parameters of interest for the algorithm. To evaluate the impact of this amount on processing time for each shader program, a series of experiments was carried with same sources of image information, as in the previous case.

The values are averaged over all shader programs with a given number of parameters of matrix type for a ten minute record. The results are shown in Fig. 3.

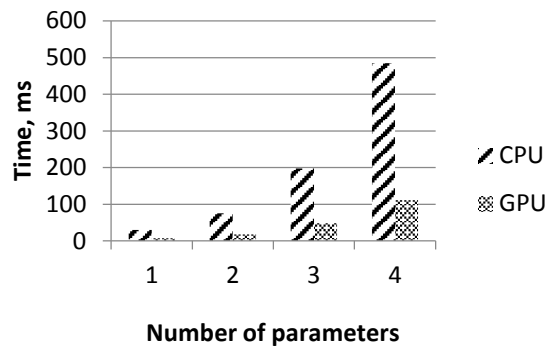


Fig. 3. Diagram of time depending on the number of parameters

Number of recognizable parameters affects their recognition duration exponentially. Speed of data processing strongly depends on the complexity of video source rendering system. However, GPGPU calculations can reduce it by 8-12 times. This allows comparison of vector video frames and subsequent profiling on the terms that are acceptable to use these methods in practice.

REFERENCES

- [1] Thorsten Scheuermann, and Justin Hensley, "Efficient histogram Generation Using Scattering on GPUs" in Proceedings of the 2007 symposium on Interactive 3D graphics and games, ACM New York, NY, USA, 2007, pp. 33-37.
- [2] N. Cornelis, and L. Van Gool, "Fast Scale Invariant Feature Detection and Matching on Programmable Graphics Hardware", IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008, pp. 1-8.
- [3] N. K. Govindaraju, E. S. Larsen, J. Gray, and D. Manocha, "A memory model for scientific algorithms on graphics processors", in Proceedings of the ACM/IEEE Conference on Supercomputing (SC'06), no. 89, NY, USA: ACM Press, 2006, pp. 6-15.
- [4] V. Podlozhnyuk, "Histogram calculation in CUDA. Technical report", NVIDIA, 2007, http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/histogram64/doc/histogram.pdf
- [5] Ramtin Shams, and R. A. Kennedy, "Efficient Histogram Algorithms for NVIDIA CUDA Compatible Devices", Australia, Gold Coast, ICSPCS, 2007. pp. 418-422.
- [6] Cedric Nugteren, Gert-Jan van den Braak, Henk Corporaal, and Bart Mesman, "High Performance Predictable Histogramming on GPUs: Exploring and Evaluating Algorithm Trade-offs" in Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, NY, USA: ACM New York, 2011. pp. 1-9.
- [7] O. Fluck, S. Aharon, D. Cremers, and M. Rousson, "GPU histogram computation", in ACM SIGGRAPH 2006 Research posters, SIGGRAPH '06. ACM, 2006, p. 53.
- [8] Adityo Mahardito, Adang Suhendra, and Deni Tri Hasta, "Optimizing Parallel Reduction In Cuda To Reach GPU Peak Performance", in Proceedings of The Second International Workshop on Open source and Open Content WOSOC 2010, Indonesia, Depok.: Gunadarma University, 2010, pp. 48-57.
- [9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "Speeded-Up Robust Features (SURF)", Computer Vision and Image Understanding, vol. 110, New York, USA, 2008, pp. 346-359.
- [10] Timothy B. Terriberry, Lindley M. French, and John Helmsen. "GPU Accelerating Speeded-Up Robust Features", in Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization and Transmission, Georgia Institute of Technology, Atlanta, GA, USA, 2008. pp. 355-362.

Two-step Harmonious Melody Generator

Sofya Latkina

*#Software Engineering Department, Faculty of Computer Science,
National Research University Higher School of Economics
Russia, Moscow, 20 Myasnitskaya, 20*

[¹latkina.sofya@gmail.com](mailto:latkina.sofya@gmail.com)

salatkina@edu.hse.com

Abstract — This paper describes the problem of creating computer music without interruption of humankind in generating algorithm working. It contains review of existing solutions, description of their key features and brand new approach explanation, which lets generate music in non-traditional way and uses research achievements from another scientific field.

Keywords — music creating, algorithm, computer music, harmony, 3 dimensional, cybernetics, data analysis, random, generation

I. INTRODUCTION

A. Problematic area

As man develops and explores new levels of technological progress, appearance of high-speed computers broadened the range of non-mathematical problems, allowing algorithmic description and simulation at the information level of processes related to human creative activity. The computer as a technological unit has evolved from a simple calculator to a distributed system, supporting million non-recurring processes, sophisticated mechanism of artificial intelligence emulation, or a life support equipment. Essentially, Hi-Tech invades to every sphere of human activity, even to the complicated ones, related to nonlinear thinking and abstract mindset, like an art.

Specifically, music as a piece of art is not a trivial product for being produced by computer as it requires integrity, variability, and harmoniousness. Generally, discussions about the definition of music are reduced to two contradictory definitions: "Music is the language of our emotions", and "Music — a calculation of the mind, unsuspecting of these calculations" (Leibniz). Music is composed of elements and refined sequences of them that affects listeners' perception and sensations. Moreover, man is able to differ melodic elements depending on their "pleasantness" of exposure. This acoustic "pleasantness" is easily amenable to analysis and explanation, while the simulation of these effects and machinery reproduction is still under investigation.

B. Background observation

The first attempts to use the information approach in the study of musical art are related to the achievements of classic statistical information theory. This theory in the classical Shannon version has had a purely technical orientation. It was designed for communications and was almost bounded by this area. However, in 1950-60 it began to rapidly penetrate into various research area.

One of the first statistical study of music theory with the methods of information theory was undertaken in 1956 by American scientist Robert K. Pinkerton. In the article "Information Theory and Melody" [1] he questioned what makes a melody attractive; he discussed the issue in mathematical term. For that Pinkerton analyzed information theory in popular American tunes and children's songs to determine the probability of individual notes and paired combinations appearance. Moreover, he calculated the entropy per one note and the information redundancy. Basing on the probability of two consecutive notes with a help of random selection, he was able to make few tunes, similar to analyzed ones. Unfortunately, most of them seemed to be monotonous and not attractive enough. This fact allowed scientist to admit that not only every single note conveys a certain amount of information but also that for obtaining "attractive" tunes some redundancy is needed.

The same goal (making up new tunes by probabilistic selection) has become the basis of the study, named "The experiment in music song" [2], which was implemented in 1957 in the laboratory of computers at Harvard University. Several scientists analyzed excerpts from 37 hymns of different composers and epochs. Scientists used computer equipment for counting frequencies of all the individual elements as well as all combinations of two, three, and so on up to eight neighboring elements. But the discovery of statistical regularities was only the initial stage of their study. Basing on these results, scientists have tried to build a computer model for the creation of music. The resulting table of sounds probability and their connections have been used for the synthesis of melodies via a random process. In total, scientists have made about 6,000 attempts of a synthesis, and created approximately 600 hymns. It should be noted that the calculations in this study were made without direct bearing on the mathematical apparatus of information theory. Incidentally, this is indirect evidence that the necessary and sufficient sought computational results can be obtained, limiting the methods of probability theory.

Since then appeared a substantial amount of applications and systems that challenge computing technology in music composition. As the development in this area has started, many new theories and concepts appeared. Human taught computer basic aspects of music: sound synthesis, digital signal processing, sound design, sonic diffusion, acoustics, and psychoacoustics. The complex path of computer music investigation can be traced back to the origins of electronic

music creation, and the first innovations and experiments with electronic instruments at the turn of the XX century.

There is a big selection of systems that provide digital music. Some of them require human interruption to a greater extent, like those ones developed in 50s (CSIRAC, playing Colonel Bogey March [3], Ferranti Mark 1 computer (MUSIC I [5]), the biggest achievement of which were the incipience of algorithmic composition programs beyond rote playback. Some of concepts are more independent, like TOSBAC computer [6] which caused resonance in the area and became an origin of computer music carried out for commercial purposes in popular music (this has led to the use of computers in widespread in the editing of pop songs). For the current moment, the terms of “computer music” or “computer-generated music” are related to any music which uses computers in its composition (that implies a kind of music which cannot be created without the use of computers).

Nowadays, intensive researches in the field of computer music creation are continuously carried out. Several mighty organizations are engaged (ICMA ¹, IRCAM ², SEAMUS ³) and some institutions of higher learning also.

Besides scientific studies, the specialists and composers have also created some software solutions, which can be considered as basic concepts: topical for today and for contemporary computer music concepts.

In the current context it is worth to mention widely known numerous experiments and studies of R. H. Zaripov. For simulating the process of composing music, he has created several programs, which were based on different principles. At first he used the principle of synthesizing music from individual sounds; next he subdued an algorithm to certain structural, rhythmic, of pitch and harmonic laws [7, pp. 90-118; 79]; then he treated musical pattern as well as poetic text [7, pp. 119-140]; finally he approached borrowing the most common melodic turns in intonation in order to create similar melody [8]. Furthermore, it was established program-harmonizer, which imitates the process of solving the problem of melodies harmonization by students of music schools [7, pp. 141-175].

C. Composition

The method of new melodies composing plays vital role in concepts of computer music creation. Musical composition simultaneously relates to the notion of an original piece of music, to the structure of a musical piece, to the process of creating some new melody. In general, the composition consists of manipulation of each aspect of music (harmony, melody, form, rhythm, and timbre). When computer music is created, it usually means that new musical notation appeared as a result of improvisation or selection and completion of patterns but more often as a result of sophisticated algorithm operating.

¹ The International Computer Music Association

² Institut de Recherche et Coordination Acoustique/Musique (France)

³ Society for Electro Acoustic Music in the United States

There can be roughly defined several common types of algorithms, basing on which exact instruments are used in a process of composing:

- Mathematical models,
- Knowledge-based systems,
- Grammars,
- Evolutionary methods,
- Systems that learn,
- Hybrid systems.

The specificity of each type is clearly implied by its name.

Currently, intensive and promising researches are undertaken in the fields of generative and evolutionary music. Also the improvisation as an efficient method of computer music making can be highlighted.

1) *Generative music*: The original term was popularized by Brian Eno, English composer and well-known innovator in ambient music; it implies the music, which is created by a computer and appears to be constantly changing and different. For an explicit indication that some clarification is needed; according to R. Wooller [9], there are four primary interpretations of generative music:

- **Linguistic/structural**: Music made up using analytic theoretical constructs, explicit as much as it is needed for generating structurally coherent material. The roots can be traced back to the generative principles in grammar of language and music, where generative instead refers to mathematical recursive tree structure.
- **Interactive/behavioural**: Music created by a system component with no discernible musical inputs, i.e., “not transformational”. Example: engine Koan, developed by SSEYO.
- **Creative/procedural**: Music composed as a result of processes set which are designed and/or set in motion by the composer. Examples of result: “In C” by Terry Riley and “Its gonna rain” by Steve Reich.
- **Biological/emergent**: Music which can be defined as non-deterministic, revolved around the idea of using “farming” parameters for creating different variation of sounds (such as wind chimes). Example: collaborative electronic noise music symphony “Viral symphony” by Joseph Nechvatal.

2) *Evolutionary music*: This type of computer music is created using an evolutionary algorithm (a subset of evolutionary computation that is based on mechanisms of biological evolution, such as reproduction, mutation, recombination, and selection, and is aimed at optimization of processed essence). The whole process initiates with a set of individuals which produce audio (a piece of music, or melody, or loop): these can be generated randomly or produced by human mind. Then, through the repetitious taking steps of computation, this population becomes optimized, more sounding like a piece of customary music. As it is quite a complicated task for a computer to determine how exactly piece of art is sounding, typically the user or audience is used

as fitness function (objective function that is used as a single figure of merit) of interactive evolutionary algorithm. Additionally, methods of evolutionary processing are commonly applied to harmonization and accompaniment tasks.

It is worth noting, that research in the field of automated measures of musical quality, which can be implemented by a simple computer, is also conducted nowadays. Example: NEUROGEN software uses a genetic algorithm for producing and combining musical fragments and a set of neural networks (initial population of individuals is based of real music) [10].

3) *Computer-Aided Algorithmic Composition*: The most common method of machine improvisation is a recombination of different musical phrases. As the resulting computer music has to be credible and nice-sounding, machine learning and pattern matching algorithms are inevitably used. That normally causes creating of variations “in the style” of original melody or pieces of music.

Modelling the particular style is a complicated objective, it requires statistical handling, big data to some extent. The algorithm can use musical surface to distinguish key stylistic features. This approach uses terms of pattern dictionaries for subsequent generating the new audio. This long musical tradition was started on 60s with Markov chains and stochastic processes. Nowadays lossless data compression for incremental parsing, pattern searching, prediction suffix tree and other new methods of data processing were added.

The factor of convenient usage of natural interface, where the musician has no need for coding musical algorithms, leads to prevalence of such systems in live performances.

Example: OMax, developed in IRCAM.

D. Main purposes and objectives

It should be emphasized that the researches in the field of computer music creating and different generative, evolutionary, or improvisation approaches, the development of the original algorithm, and the grasp of the concept of intuitive human-computer interaction, which will allow to manage the process of music creating, pursue the same goal. The primary aim of the entire project is to create computer music generator which will be able to create melodies according to the settings, specified by user, but without actual interruption of user to the generation of melodic pattern.

Undoubtedly, it is vital to perform specific objectives in order to reach the goal of the research. It seems to be important to clarify them in detail. The first objective will be accomplished by inventing an algorithm of computer music generating. Inevitably, it will be based on existing methodologies (generative, evolutionary), but it also has to be sharpened by the principle of flexibility and ability of changing according to adjustments, made by user. Next objective is to implement software shell, which will satisfy potential user and allow to manipulate melody relatively effortless and without necessity of code changing. Finally, output methods have to be elaborated: the way of music

sounding is one of the most important things in the sphere of computer music creating.

Essentially, there is can't be any need to verify and prove what way of music creating is better, more efficient, of aesthetical: the traditional one, or the innovative variations. The interlinear mission of the whole work is to extend musical thinking or composition practice which is current computer-music practice.

II. METHODOLOGY

The destination of software which is able to produce music is to create the successions of musical tones that can be perceived as melodies, pieces of art. Considering a definition given by Alexander I. Ringer, “melody” is a pitched sounds arranged in musical time in accordance with given cultural conventions and constraints [11]. It can be noted that in some cultures rhythmic considerations may take precedence over melodic expression, so the cultural and regional context largely determines what exactly a human accepts as music. For example, Chinese and European perception of music differs a lot; this is due to many factors, in particular: the time of development of the national understanding of musical composition.

According to ancient Chinese encyclopedic works *Lüshi Chunqiu*, the scale has to contain twelve tones. The situation differs for European music, which is younger and fully aligned with the Well-Tempered Clavier of Bach. Current paper corresponds to the European scale and standards of Western music. In this concept a pitch space includes octaves sized 12 semitones — this specific distance reflects physical distance on keyboard instruments, orthographical distance in Western musical notation, and musical distance as measured in psychological experiments [9].

A. Tones and scale

Tones, which construct a melody, equal to the sum of two semitones and hence referred to as a ‘whole tone’, usually perceived as a major 2nd; in equal temperament, the sixth part of an octave. As it is defined for European scale, the semitone seems to be the ration of the frequencies as 1 to the 12th degree of 2. Thus, the tone of particular note can be identified with function: $f(x_i) = \sqrt[12]{2} * f(x_{i-1})$, where x_i is a current note and x_{i-1} — the previous one.

Tones are used in musical theory for calculating intervals, which inevitably appear “between” every two notes. Literally speaking, this circumstance affects a lot on how a person perceives a melody, whether he likes it or not, recognizes as music or not.

The set of intervals is restricted, each of them has two vital characteristics: the amount of semitones and harmoniousness. Shortly, mostly used intervals can be presented in the following list:

- Perfect unison, perfect octave — the best consonance;
- Perfect fourth, perfect fifth — middle consonance;
- Third (minor, major), sixth (minor, major) — imperfect consonance;

- Second (minor, major), seventh (minor, major) — sharp dissonance.

B. Harmony

According to the New Grove Dictionary of Music and Musicians, harmony can be defined as combining of notes simultaneously, to produce chords, and successively, to produce chord progressions. The term is used descriptively to denote notes and chords so combined, and also prescriptively to denote a system of structural principles governing their combination [11]. Creating a harmonic and logical melody is a sophisticated task, which is complicated by a sufficient number of rules, restrictions, and preconditions. Important mention: “logical” in this context implies symmetry of melody, adherence to pre-defined rules, compliance with the restrictions, exactly. Logical construction of melody includes controlling what next note will be, where the start and the end of melody are, at what time the next transition can be performed. Existing tools can provide the solution of these important tasks.

C. Petri nets

Once an issue of polyphony is raised, the usage of Petri nets seems to be relevant. Creating computer music becomes more complicated if second (third, fourth, etc.) voice is added. Without proper synchronization, created music will become cacophonous.

The dynamic system can model a “Conductor”: like a conductor in real life, this model manages two or more musical threads. It is necessary to keep tracking of hitting the strong bit and maintaining mode and harmony. Due to what can this monitoring be achieved?

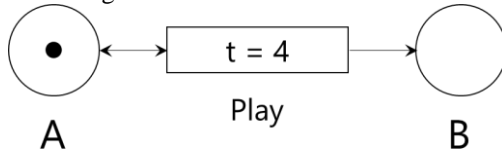


Fig. 1 Example of timed petri net

The key feature of timed Petri nets is a usage of limited execution time, which makes the transition disabled from occurring for the duration time; but it is fired immediately after becoming enabled. In the presented primitive net (see in Fig. 1) the time delay (or execution time) is 4 time units. In the initial state “Play” in enabled will therefore immediately fire, i.e., the token in A is consumed. Next there occurs a delay in 4 time units before the firing is complete and tokens are deposited into A and B. Now Play is again enabled and will again fire.

Practical application of the concept can be demonstrated on the following example (see in Fig.2): in the first bar (Bar0) only one violin plays, next the second violin joins, then the first violin sounds together with two viols, finally, all instruments play together, and in the last bar the first violin is again sounding lonely (see the information about tokens motion in table I). This example can provide representation of how actual conductor deals with four different musicians.

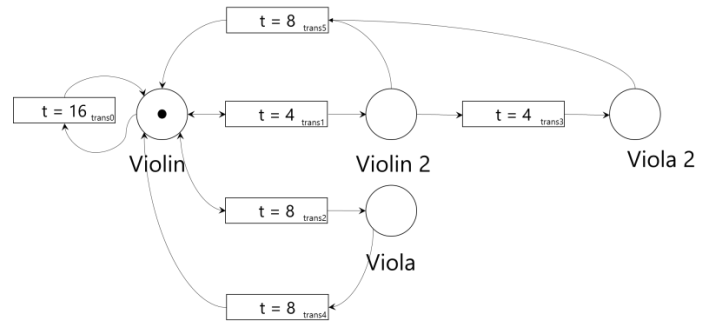


Fig. 2 Example of timed Petri net, model “Conductor”

TABLE I
CHRONOLOGY OF TOKENS MOTION

	Violin	Violin2	Viola	Viola2
<initial> (Bar0)	1	0	0	0
trans1 (Bar1)	1	1	0	0
trans2, trans3 (Bar2)	1	0	1	1
trans1 (Bar3)	1	1	1	1
trans0, trans5, trans6 (Bar4)	3	0	0	0

Within the scope of current paper only monophonic melodies will be considered; but usage of timed Petri nets stays suitable for the project, perspective.

III. TWO-STEP HARMONIOUS COMPUTER MUSIC CREATION ALGORITHM

The process of creating computer music with a melody as a resulting form can be divided in two phases: first, computer constructs durational pattern of melody, then, it is filled with tones.

A. Durational pattern construction

A typical melody is a combination of pitches and rhythm. It is not essential what element of combination will be created first; in the current work it will be the rhythm.

All rhythmic units can be classified as (see in Fig. 3):

- *Metric* — even patterns, such as steady eighth notes or pulses;
- *Intrametric*—confirming patterns, such as dotted eighth-sixteenth note and swing patterns;
- *Contrametric*—non-confirming or syncopated patterns;
- *Extrametric*—irregular patterns, such as triplets.

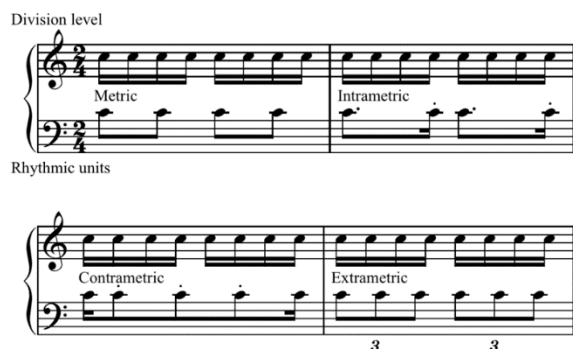


Fig. 3 Rhythmic units

The realization of each kind of rhythmic units becomes possible with a proper standardization of a variety of notes durations. In this way, for every duration (eighths, pulses) the time is given: exact amount of seconds, for which a single note with this duration sounds. This parameter (the time) can be accordingly changed if a tempo of the whole melody is changed.

By creating durational pattern, a program complies with necessary restrictions, like: an overall sum of beats doesn't exceed time (meter) signature. It also avoids syncopation for the first and last beats of pattern and adheres to the principle of symmetry.

Durational pattern of musical compositions appears to be holistic and logical if it uses principles of symmetry and repetition. Like in poems, rhythmical phrases have to alternate. By this reason, algorithm considers the amount of bars, which have to be filled with various durations, and constructs an alteration of several rhythmic patterns, just as if it comes to the rhyme in the poem. The process is organized in the following way: A, B, C, D – rhythmical phrases, the combination of several durations, overall amount of which doesn't exceed time signature. Program generates from 1 to 4 different phrases and constructs the durational pattern like a poem, using one of the six schemes (each named by similar rhyme scheme), described in Table II.

TABLE II
RHYTHM SCHEMES

Name of scheme	Phrases alternation (for 4 bars)
Alternate	A B A B
Enclosed	A B B A
Monorhyme	A A A A
Rubaiyat	A A B A
Simple 4-line	A B C B
Clerihew	A A B B

After 4 bars of durational pattern are constructed, program deals with next ones, using the same rhythmic scheme or another one.

Here is a short example of how algorithm creates durational pattern for eight bars with time signature C or $\frac{4}{4}$ in Table III (here only metric patterns are used in order to facilitate understanding).

TABLE III
EXAMPLE OF DURATIONAL PATTERN CONSTRUCTING

Rhythmic phrase 'A'	Crotchet + Quaver + Quaver + Crotchet + Crotchet
Rhythmic phrase 'B'	Quaver + Quaver + Quaver + Quaver + Quaver + Crotchet + Crotchet
Rhythmic phrase 'C'	Quaver + Crotchet + Quaver + Quaver + Crotchet + Quaver
Rhythmic phrase 'D'	Crotchet + Quaver + Quaver + Minim
Chosen scheme(-s)	Alternate (using phrases A,B) + Simple 4-line (using phrases A, D, C)
Resulting scheme	A B A B A D C D
Bar 1	
Bar 2	
Bar 3	
Bar 4	
Bar 5	
Bar 6	
Bar 7	
Bar 8	

B. Melodic pattern construction

The basis of this part of the algorithm lies in the rules of harmonic melody construction (rules will be explained further).

In mathematics, there is one key rule: a plane can be described through three points. Literally saying, the whole two-dimensional surface, a flat, that contains endless amount of points, can actually be defined by only three of them. A figure "3" has significant in a context of music creating also. Three notes form a chord, which determines vital characteristics of musical composition: whether it is major of minor, harmonious or disharmonious. As it is needed to create harmonious melodies, chords can be uses as basic elements, sequential playback of which is finally a musical canvas.

Back to the Western music: it occurs that this concept is a product of two subjects, harmony and counterpoint (voice leading). The first discipline appoints the acceptable chords, which sound simultaneously or successively. The second one connects the individual notes in a series of chords so as to

form simultaneous melodies. According to Dmitri Tymoczko, composer and music theorist, these key features “facilitate musical performance, engage explicit aesthetic norms, and enable listeners to distinguish multiple simultaneous melodies” [12].

This researcher has developed an interesting model of melody’s motion analysis. He supposed that there can be a geometric shape which can represent all possible notes and their combinations. This shape is an orbifold (see in Fig. 4) — that is the space of unordered pairs of pitch classes. The orbifold is singular at its top and bottom edges, which act like mirrors. In this way, and melody or voice leading between pairs of pitches (or pitches classes) can be associated with a path on the picture. And as it follows, consonant chords of traditional Western music can be connected by efficient voice leading, visualized on this shape. There are a lot of sophisticated nuances and features in the description of this model, which can be unclear for uninitiated reader. The most essential conclusion is that, after all necessary investigations, researcher has proved that most of famous classical melodies subject to common rules: they consist of symmetrical voice leadings, which can be easily traced with orbifold. This rule applies for canonical music, hence, it can be inversed. The aim of this part of algorithm in the current project is to use inversed rule and build a melody, basing on harmonious permutations and combinations.

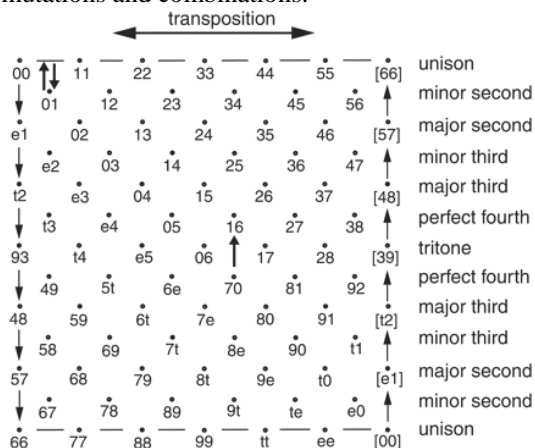


Fig. 4 Orbifold

For the particular objective simplified shape can be considered. It is a cube with eight vertices: for each pitch in octave and one for the first one of the next octave (see in Fig. 5). This cube is carried out specifically for Cdur.

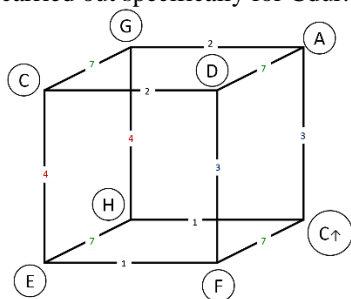


Fig. 5 Cube of pitches sequences constructing

The essence of this method is that program constructs a melody by moving along the edges: from one vertex to another. These movements are caused by the chords; program is trained to use the most harmonious ones, vary sequences, and always resolve to the tonic. How exactly does it work? It would be rational to explain the approach with an example:

0. A program has already defined durational pattern so this is not an issue anymore;
1. Program appoints C (tonic) as the first pitch;
2. Program chooses next pitch from E, F, and D. This can result in intervals: major third, quart, or major second. Program chooses F;
3. Program chooses next pitch from A, C, or C of the next octave. Only one option can result in chord, so program chooses A. End of iteration (chord is done);
4. Program chooses next pitch from H, D, or F. Program chooses H;
5. Program chooses next pitch from G, A, and D. This can result in intervals: major third, major second, or major seventh. Program chooses G;
6. Program chooses next pitch from D, E, or H. Two option can result in chord (D – to major one, E – to minor one), so program chooses D, because it deals with Cdur. End of iteration (chord is done);
7. Program chooses next pitch from A, C, or G. Program chooses A;
8. Program chooses next pitch from F, H, and D. This can result in intervals: major third, major second, or major sixth. Program chooses F;
9. Program chooses next pitch from C, C of the next octave, or A. Two option can result in chord (C and C of the next octave – both to the major ones), so program chooses C. End of iteration (chord is done);
10. And so on...

One of the key limitations for this endless process is to return to the tonic at the end of voice leading. The entropy of melodic pattern can be increased if it is allowed to move not only along edges (those ones which are drawn on the picture). But the principle has to stay unchanged: the motion considers chords and gives priority to the consonant ones.

Program picks an amount of pitches which correspond the durational pattern created earlier. At the final stage algorithm creates an object: melody, which consist of notes (objects with appropriate properties: tone and durations). This is the end of algorithm work.

IV. CONCLUSION

The problem of this paper is considered upon the problem of creating music by computer, which sounds rhythmically and harmonically and appears to be received as a complete melodic pattern without actual interruption of humankind. Its specifics is related to the consonantly sounded melodies, to simplicity of construction algorithm, and to its flexibility: in a case cancelation of some of limitation, program will provide qualitatively different piece of art, hence, the ability of

computer improvisation can become unlimited within the scope of this project while the final produce stays holistic.

REFERENCES

- [1] R. Pinkerton. "Information Theory and Melody", *Scientific American*, vol. 194. #2, pp. 77-86, 1956.
- [2] F. Brooks, A. Hopkins, P. Neumann, W. Wright. "An experiment in musical composition", *IRE Transactions on Electronic Computers*, vol. EC-6, № 3, pp. 175–182, 1957.
- [3] P. Doornbusch, *The Music of CSIRAC*, Melbourne School of Engineering, Department of Computer Science and Software Engineering, Ed. Melbourne, Australia: Common Ground, 2005.
- [4] J. Fildes, "'Oldest' computer music unveiled", *BBC News*, Dec. 2008, retrieved Dec. 4, 2013.
- [5] V. Bogdanov, *All Music Guide to Electronica: The Definitive Guide to Electronic Music*, Russia: Backbeat Books, 2001
- [6] T. Shimazu, "The History of Electronic and Computer Music in Japan: Significant Composers and Their Works", *Leonardo Music Journal (MIT Press)*, vol. 4, pp. 102-106, 1994.
- [7] R. Zaripov, *Cybernetics and music*, Moscow, Russia: Nauka (1971) (in Russian).
- [8] R. Zaripov, "The production system in the music", *Proceedings of the Academy of Sciences of the USSR. Technical Cybernetics*, v. 2, pp. 207-216, 1987 (in Russian).
- [9] R. Wooller, A. Brown, "A framework for comparing algorithmic music systems", in *Symposium on Generative Arts Practice (GAP)*, 2005.
- [10] P. M. Gibson, J. A. Byrne (1991) NEUROGEN, musical composition using genetic algorithms and cooperating neural networks. [Online]. Available: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=140338>.
- [11] *The New Grove Dictionary of Music and Musicians*, 2nd ed., S. Sadie, Ed., J. Tyrrell, Ed. Oxford, UK: Oxford University Press, 2004.
- [12] D. Tymoczko (2006) The Geometry of Musical Chords. [Online]. Available: <http://www.sciencemag.org/content/313/5783/72.full>.
- [13] D. G. Loy, *The Music Machine: Selected Readings from Computer Music Journal*, Roads, Curtis, Ed. Cambridge, USA: MIT Press, 1992.
- [14] B. Varga, U. Dimen, and E. Loparitz, *Language, music, mathematics*, Moscow, Russia: Mir (1981) (in Russian).

A Crowdsourcing Engine for Mechanized Labor

Dmitry Ustalov

N.N. Krasovskii Institute of Mathematics and Mechanics

Ural Branch of the Russian Academy of Sciences

16 Sofia Kovalevskaya st., Yekaterinburg, Russia

Email: dau@imm.uran.ru

Abstract—Crowdsourcing is an established approach for producing and analyzing data that can be represented as a human-assisted computation system. This paper presents a crowdsourcing engine that makes it possible to run a highly customizable hosted crowdsourcing platform controlling the entire annotation process including such elements as task allocation, worker ranking and result aggregation. The approach and the implementation have been described, and the conducted experiment shows promising preliminary results.

Keywords—crowdsourcing engine, mechanized labor, human-assisted computations, task allocation, worker ranking, answer aggregation.

I. INTRODUCTION

Nowadays, crowdsourcing is a popular and a very practical approach for producing and analyzing data, solving complex problems that can be splitted into many simple and verifiable tasks, etc. Amazon's MTurk¹, a well known labor marketplace, promotes crowdsourcing as the *artificial artificial intelligence*.

In the *mechanized labor* genre of crowdsourcing, a requester submits a set of tasks that are solved by the crowd workers on the specialized platform. Usually, the workers receive micropayments for their performance, hence, it is of high interest to reach the happy medium between the cost and the quality. The work, as described in this paper, makes the following contributions: 1) it presents a survey on crowdsourcing control approaches and 2) presents an engine for controlling a crowdsourcing process.

The rest of this paper is organized as follows. Section II reviews the related work. Section III defines the problem of lacking the control software for crowdsourcing. Section IV presents a two-layer approach for crowdsourcing applications separating the engine from the end-user application. Section V describes the implementation of such an engine. Section VI briefly tests the present system. Section VII concludes with final remarks and directions for the future work.

II. RELATED WORK

There are several approaches for controlling the entire crowdsourcing process.

Whitehill et al. proposed the *GLAD*² model that, for the first time, connects such variables as task difficulty, worker experience and answer reliability for image annotation [1].

Bernstein et al. created the *Soylent* word processor, which automatically submits text formatting and rewriting tasks to the crowd on MTurk [2]. The paper also introduces the *Find-Fix-Verify* workflow, which had highly influenced many other researchers in this field of study.

Demartini, Difallah & Cudré-Mauroux developed *ZenCrowd*, another popular approach for controlling crowdsourcing, which was originally designed for mapping the natural language entities to the Linked Open Data [3]. ZenCrowd is based on the EM-algorithm and deploys the tasks to MTurk.

The idea of providing an integrated framework for a crowdsourcing process is not novel and has been addressed by many authors both in academia and the industry, e.g. WebAnno [4], OpenCorpora [5] and Yet Another RussNet [6].

However, the mentioned products are problem-specific and using them for crowdsourcing different tasks may be non-trivial. Moreover, that software do often force the only possible approach for controlling the process of crowdsourcing, which in some cases may result in suboptimal performance.

A. Task Allocation

Lee, Park & Park created a dynamic programming method for task allocation among workers showing that consideration of worker's expertise increases the output quality [7].

Yuen, King & Leung used probabilistic matrix factorization to allocate tasks in the similar manner that recommender systems do [8].

Karger, Oh & Shah proposed a budget-optimal task allocation algorithm inspired by belief propagation and low-rank matrix approximation being suitable for inferring correct answers from those submitted by the workers [9].

B. Worker Ranking

Welinder & Perona presented an online algorithm for estimating annotator parameters that requires expert annotations to assess the performance of the workers [10].

Difallah, Demartini & Cudré-Mauroux used social network profiles for determining the worker interests and preferences in order to personalize task allocation [11].

Daltayanni, de Alfaro & Papadimitriou developed the *WorkerRank* algorithm for estimating the probability of getting a job on the *oDesk* online labor marketplace utilizing employer implicit judgements [12].

¹<http://mturk.com/>

²<http://mplab.ucsd.edu/~jake/>

C. Answer Aggregation

The answers are often aggregated with majority voting, which is highly efficient for small number of annotators per question [9]. Some works use a fixed number of answers to aggregate [5].

Sheshadri & Lease released SQUARE³, a Java library containing implementations of various consensus methods for crowdsourcing [13], i.e. such methods as ZenCrowd [3], majority voting, etc.

Meyer et al. developed *DKPro Statistics*⁴ implementing various popular statistical agreement, correlation and significance analysis methods that can be internally used in answer aggregation methods [14].

D. Cost Optimization

Satzger et al. presented an auction-based approach for crowdsourcing allowing workers to place bids on relevant tasks and receive payments for their completion [15].

Gao & Parameswaran proposed algorithms to set and vary task completion rewards over time in order to meet the budget constraints through the use of Markov decision processes [16].

Tran-Thanh et al. developed the Budgeteer algorithm for crowdsourcing complex workflows under budget constraints that involves inter-dependent micro-tasks [17].

III. PROBLEM

Hosseini et al. defines the four pillars of crowdsourcing making it possible to represent the crowdsourcing system C as the following quadruple [18]:

$$C = (W, R, T, P) \quad (1)$$

Here, W is the set of workers who benefit from their participation in the process C , R is the task requester who benefits from the crowd work deliverables, T is the set of human intelligence tasks provided by the requester R , and P is the crowdsourcing platform that connects these elements.

Unfortunately, there is no open and customizable software for controlling C . This problem is highly topical since using MTurk, the largest crowdsourcing platform, is not possible outside the U.S. making it interesting to develop an independent substitution that can be hosted.

IV. APPROACH

The reference model of a typical mechanized labor crowdsourcing process is present at Fig. 1 and consists of the following steps repeated until either convergence is achieved or the requester stops the process:

- 1) a *worker* requests a *task* from the *system*,
- 2) the *system* allocates a *task* for that *worker*,
- 3) the *worker* submits an *answer* for that *task*,
- 4) the *system* receives and aggregates the *answer*,
- 5) the *system* updates the *worker* and *task* parameters.

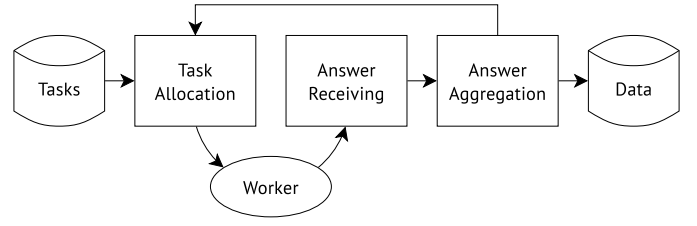


Fig. 1. Reference Model

A. Use Case Diagram

Modern recommender systems like PredictionIO⁵ and metric optimization tools like MOE⁶ separate the *application* layer from the *engine* layer to simplify integration into the existent systems. In crowdsourcing, it is possible to separate the worker annotation interface (the application) and the crowdsourcing control system (the engine) for the same reason.

The use case diagram present at Fig. 2 shows two actors—the requester and the application—interacting with the engine. The application works with the engine through the specialized programming interface (API) and the requester works with the engine using the specialized graphical user interface (GUI).

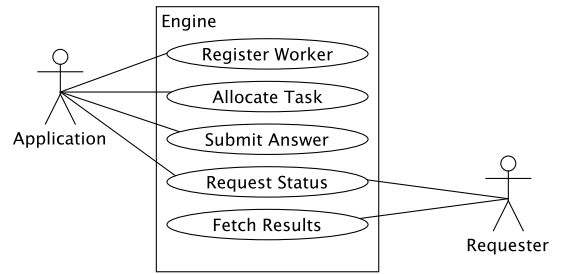


Fig. 2. UML Use Case Diagram

B. Sequence Diagram

The sequence diagram at Fig. 3 shows the interaction between those elements: a worker uses the end-user application that is connected to the engine that actually controls the process and provides the application with the appropriate data.

V. IMPLEMENTATION

The proposed system is implemented in the Java programming language as a RESTful Web Service through the use of such APIs as JAX-RS⁷ and JDBI⁸ within the Dropwizard⁹ framework. The primary data storage is PostgreSQL¹⁰, a popular open source object-relational database.

A. Class Diagram

The class diagram at Fig. 4 represents the crowdsourcing system as according to the equation 1. The `Process` class

⁵<http://prediction.io/>

⁶<https://github.com/Yelp/MOE>

⁷<https://jcp.org/en/jsr/detail?id=339>

⁸<http://jdbi.org/>

⁹<http://dropwizard.io/>

¹⁰<http://www.postgresql.org/>

³<http://ir.ischool.utexas.edu/square/>

⁴<https://code.google.com/p/dkpro-statistics/>

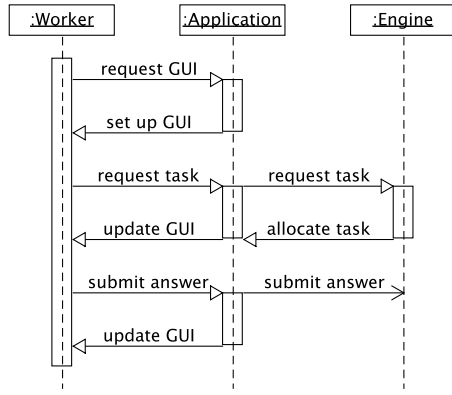


Fig. 3. UML Sequence Diagram

defines a system C and specifies how its elements W , T and A should be processed by the corresponding implementations of the abstract `Processor` class.

Particularly, an actual processor inherits that abstract class and implements one or many of the following interfaces: `WorkerRanker`, `TaskAllocator`, `AnswerAggregator`. The reason for that is the dependency uncertainty of each particular processor implementation that has been approached by the dependency injection mechanism¹¹.

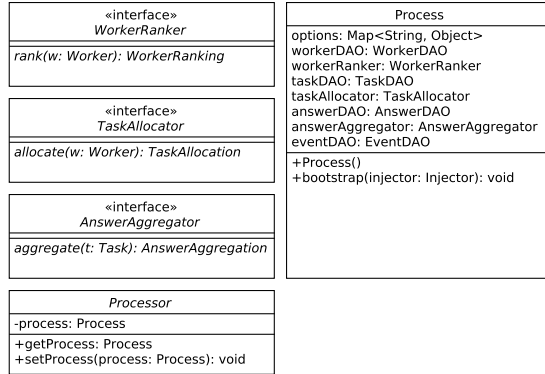


Fig. 4. UML Class Diagram

For example, a processor for majority voting, which is a popular approach for answer aggregation, should extend the `Processor` class and provide the implementation of the `aggregate` method inherited from the `AnswerAggregator` interface. In order to access the answers stored in the database, the corresponding data access object—`AnswerDAO`—should be injected.

On startup, the application configures itself with the provided configuration files and database entries, sets up the Guice¹² dependency injector and creates an instance of each defined process. Then, the application calls the `bootstrap`

method of each process, which initializes the processors specified in the process configuration. Finally, these resources are getting exposed by the RESTful API.

B. Package Diagram

The system is composed of several packages responsible for its functionality. Since that the Dropwizard framework is used, the most of boilerplate code is already included in the framework. However, such a sophisticated initialization requires additional middleware resulting in the package hierarchy represented at Fig. 5 detailed in Table I.

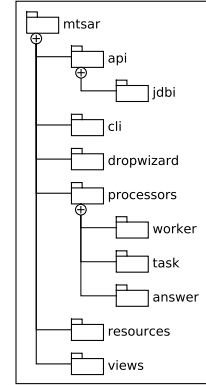


Fig. 5. UML Package Diagram

TABLE I. PACKAGES

Package	Description
mtsar	Utility classes useful to avoid the code repetition.
mtsar.api	Entity representations.
mtsar.api.jdbi	JDBI's data access objects and object mappers.
mtsar.cli	Command-line tools for maintenance and evaluation tasks.
mtsar.dropwizard	Middleware for Dropwizard.
mtsar.processors	Actual implementations of the methods for controlling workers, tasks, answers.
mtsar.resources	Resources exposed by the RESTful API.
mtsar.views	View models used by the GUI.

VI. EVALUATION

The system functionality is tested using JUnit¹³. At the present moment, only classes contained in the `mtsar.processors` and `mtsar.resources` packages are provided with the appropriate unit tests. The continuous integration practice is followed by triggering a build on Travis CI¹⁴ for each change to ensure that all the unit tests have been successfully passed.

In order to make sure the system works, the RUSSE¹⁵ crowdsourced dataset has been used. The `russe` process has been configured to use the `zero` worker ranker that simply ranks any worker with zero rank, `inverse count` task allocator that allocates the task with the lowest number of available answers, and the `majority voting` answer aggregator (Fig. 6). Then, the workers, tasks and answers stored in this dataset have been submitted into the system via the RESTful API and the conducted experiment showed that no data have been

¹¹<https://jcp.org/en/jsr/detail?id=330>

¹²<https://github.com/google/guice>

¹³<http://junit.org/>

¹⁴<https://travis-ci.org/>

¹⁵<http://russe.nlpub.ru/>

Process "russe"

Key	Value	Action
workerCount	280	Details
workerRanker	mtsar.processors.worker.ZeroRanker	
taskCount	398	Details
taskAllocator	mtsar.processors.task.InverseCountAllocator	
answerCount	4200	Details
answerAggregator	mtsar.processors.answer.MajorityVoting	

Additional Options

Key	Value
i No additional options found.	

Dashboard Processes GitHub

Mechanical Bar

Fig. 6. Graphical User Interface

lost during this activity and the engine does allocate tasks and aggregate answers correctly w.r.t. the chosen processors.

VII. CONCLUSION

In this study, a crowdsourcing engine for mechanized labor has been presented and described among the used approach and its implementation. Despite the conducted experiment showing promising preliminary results, there are the following reasons for the further work.

Firstly, it is necessary to conduct a field study, which was not possible due to the lack of time. Secondly, it is necessary to integrate state of the art methods for worker ranking, task allocation and answer aggregation into the engine to provide a requester with the best annotation quality at the lowest cost. Finally, it may be useful to extend the engine API and GUI in order to make it more convenient and user-friendly.

The source code of the system is released on GitHub¹⁶ under the Apache License. The documentation in Russian is available on NLPub¹⁷.

ACKNOWLEDGMENT

This work is supported by the Russian Foundation for the Humanities, project no. 13-04-12020 "New Open Electronic Thesaurus for Russian". The author is grateful to the anonymous referees who offered useful comments on the present paper.

REFERENCES

- [1] J. Whitehill, P. Ruvolo, T. Wu, J. Bergsma, and J. Movellan, "Whose Vote Should Count More: Optimal Integration of Labels from Labelers of Unknown Expertise," in *Advances in Neural Information Processing Systems* 22. Curran Associates, Inc., 2009, pp. 2035–2043.
- [2] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: A word processor with a crowd inside," in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '10. New York, NY, USA: ACM, 2010, pp. 313–322.

- [3] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux, "ZenCrowd: Leveraging Probabilistic Reasoning and Crowdsourcing Techniques for Large-Scale Entity Linking," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 469–478.
- [4] S. M. Yimam, I. Gurevych, R. E. de Castilho, and C. Biemann, "WebAnno: A Flexible, Web-based and Visually Supported System for Distributed Annotations," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Sofia, Bulgaria: Association for Computational Linguistics, 2013, pp. 1–6.
- [5] V. Bocharov, S. Alexeeva, D. Granovsky, E. Protopopova, M. Stepanova, and A. Surikov, "Crowdsourcing morphological annotation," in *Computational Linguistics and Intellectual Technologies: papers from the Annual conference "Dialogue"*, vol. 12 (19). Moscow: RSUH, 2013, pp. 109–124.
- [6] P. Braslavski, D. Ustalov, and M. Mukhin, "A Spinning Wheel for YARN: User Interface for a Crowdsourced Thesaurus," in *Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics*. Gothenburg, Sweden: Association for Computational Linguistics, 2014, pp. 101–104.
- [7] S. Lee, S. Park, and S. Park, "A Quality Enhancement of Crowdsourcing based on Quality Evaluation and User-Level Task Assignment Framework," in *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*. IEEE, 2014, pp. 60–65.
- [8] M.-C. Yuen, I. King, and K.-S. Leung, "TaskRec: A Task Recommendation Framework in Crowdsourcing Systems," *Neural Processing Letters*, pp. 1–16, 2014.
- [9] D. R. Karger, S. Oh, and D. Shah, "Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems," *Operations Research*, vol. 62, no. 1, pp. 1–24, 2014.
- [10] P. Welinder and P. Perona, "Online crowdsourcing: Rating annotators and obtaining cost-effective labels," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2010, pp. 25–32.
- [11] D. E. Difallah, G. Demartini, and P. Cudré-Mauroux, "Pick-A-Crowd: Tell Me What You Like, and I'll Tell You What to Do," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pp. 367–374.
- [12] M. Daltayanni, L. de Alfaro, and P. Papadimitriou, "WorkerRank: Using Employer Implicit Judgements to Infer Worker Reputation," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, ser. WSDM '15. New York, NY, USA: ACM, 2015, pp. 263–272.
- [13] A. Sheshadri and M. Lease, "SQUARE: A Benchmark for Research on Computing Crowd Consensus," in *First AAAI Conference on Human Computation and Crowdsourcing*, 2013, pp. 156–164.
- [14] C. M. Meyer, M. Mieskes, C. Stab, and I. Gurevych, "DKPro Agreement: An Open-Source Java Library for Measuring Inter-Rater Agreement," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, 2014, pp. 105–109.
- [15] B. Satzger, H. Psailer, D. Schall, and S. Dustdar, "Auction-based crowdsourcing supporting skill management," *Information Systems*, vol. 38, no. 4, pp. 547–560, 2013.
- [16] Y. Gao and A. Parameswaran, "Finish Them!: Pricing Algorithms for Human Computation," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, 2014.
- [17] L. Tran-Thanh, T. D. Huynh, A. Rosenfeld, S. D. Ramchurn, and N. R. Jennings, "Crowdsourcing Complex Workflows under Budget Constraints," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*. AAAI Press, 2015, pp. 1298–1304.
- [18] M. Hosseini, K. Phalp, J. Taylor, and R. Ali, "The Four Pillars of Crowdsourcing: a Reference Model," in *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*, 2014, pp. 1–12.

¹⁶<https://github.com/dustalov/mtsar>

¹⁷<https://nlpub.ru/MTsar>

On the Implementation of a Formal Method for Verification of Scalable Cache Coherent Systems

Vladimir Burenkov

Bauman Moscow State Technical University

MCST

Moscow, Russian Federation

burenkov_v@mcst.ru

Abstract—This article analyzes existing methods of verification of cache coherence protocols of scalable systems. Based on the research literature, the paper describes a method of formal parameterized verification of safety properties of cache coherence protocols. The paper proposes a design of a verification system for cache coherence protocols. The article analyzes the method in terms of development and examination of the corresponding Promela model of the German cache coherence protocol and discusses extension and automation of the method needed to adapt it to verification challenges of the Elbrus microprocessors.

Keywords—formal verification; model checking; deductive verification; cache coherence protocol; Elbrus

I. INTRODUCTION

Modern microprocessor systems are scalable – the number of cores per chip increases and chips are combined into clusters. Each processor of the system has access to the shared address space. However, memory is physically distributed among the processors in order to increase the bandwidth and reduce the latency to local memory. Thus, access to the local memory is faster than access to the remote memory. To decrease the memory bandwidth demands of a processor, processors are equipped with multilevel caches. Caching of shared data introduces the problem of cache coherence.

To solve the problem, computer architects often use hardware mechanisms that implement cache coherence protocols. Concurrent work of many hardware devices (for example, cache and main memory controllers), which exchange information in accordance with a cache coherence protocol, results in a colossal size of the protocol's state space. This, in turn, makes verification of cache coherence protocols an extremely hard task.

To work out the problem, scientists have been conducting research in the direction of formal methods for the past few decades and achieved a level of success. However, scalable verification is still an issue.

Scalability leads to the need for formal verification methods that are capable of adapting to it. As the size of systems increases, the fully automated method of model checking reaches its limits and can no longer be used due to the state space explosion problem.

As a rule, existing formal approaches to verification are either inapplicable to industrial-strength microprocessor systems or require an enormous amount of manual work.

II. PRIMARY VERIFICATION METHODS

Formal methods provide a mathematical proof of the correspondence between a model of the object under verification and the object's specification, that is, a set of properties it is supposed to satisfy. A mathematical model of reactive systems – and cache coherence protocols are examples of reactive systems – that allows to systematically represent systems components, their coordination and interaction, is a transition system [1].

The main approaches to formal verification are model checking and deductive verification.

The method of *model checking* [2] systematically explores the finite state space of the protocol under verification by means of specific algorithms. The advantages of model checking are full automation and generation of counterexamples that help us find the sources of bugs. The main disadvantage is the state space explosion problem. Modern cache coherence protocols have too many states for an effective state space inspection to be feasible.

Let us consider verification of safety properties, which are described by linear temporal logic (LTL) formula Gp , where p is an assertion – a formula constructed by applying logical connectives to variables of the model. If the assertion is true in each state of the model, then p is an invariant of the model. According to the method of *deductive verification*, in order to prove Gp , it is necessary to develop an auxiliary assertion φ , which is an over-approximation of the state space, and then show that φ implies p (i.e., that φ is stronger than p). The method is based on the following inference rule [1]:

- | | |
|--|------|
| I1. φ is true in the initial states of the model | |
| I2. All transitions preserve φ | |
| I3. $\varphi \rightarrow p$ | |
| <hr/> | |
| | Gp |

An assertion φ is called *inductive* if it satisfies the premises I1 and I2. An inductive assertion is always an over-approximation of the set of reachable states. If p is an invariant of the system under verification, then there always exists an inductive assertion φ stronger than p [1]. The initial assertion p is rarely inductive. As a rule, the verification engineer must develop an auxiliary assertion and check the validity of the premises I1-I3.

Deductive verification allows us to work with systems with infinite number of states. Theorem provers assist in using formal logic for reasoning about mathematical objects. Popular tools are ACL2, PVS, Isabelle. The underlying logics of theorem provers vary substantially. However, all theorem provers support rich and expressive logics. In general, expressiveness of a logic leads to its undecidability. That means that there is no automatic procedure that, given a formula, can always determine if there exists a derivation of the formula in the logic. The use of theorem proving presumes interaction with an expert user and is a complicated creative process. When the theorem prover cannot find the derivation of a formula given a proof outline, it is very hard to find the actual bug in the system under verification.

Reference [3] describes the experience of using the PVS theorem prover for parameterized verification of the FLASH cache coherence protocol. During the proof construction, authors manually looked for candidates for inductive assertions many times. When they failed to prove their inductiveness, they analyzed the reasons for that and devised additional conditions that transformed the assertion into an inductive one. This process is extremely laborious, which is why methods that are solely based on theorem proving can only find a limited usage in verification of cache coherence protocols.

III. VERIFICATION METHODS FOR SCALABLE SYSTEMS

Development of verification methods for scalable systems may be carried on in several directions: 1) improvement of methods based on model checking; 2) improvement of methods based on deductive verification; 3) combination of the methods from the first and the second groups.

Methods of verification of cache coherence protocols deployed in industrial-strength microprocessor systems must satisfy a number of requirements: 1) possibility of conducting verification in a reasonable amount of time; 2) high level of automation; 3) ability to provide information about sources of bugs.

Model checking or deductive verification on their own do not meet these needs. Consequently, building a general infrastructure that would combine and further develop methods of model checking and deductive verification seems to be the most promising approach to verification of scalable systems.

IV. ABSTRACTION AND COMPOSITIONAL MODEL CHECKING

The main approaches allowing the application of model checking to verification of scalable systems are abstract model

checking and compositional verification [2]. Abstraction methods diminish the number of states of the model under verification and preserve the properties of interest at the same time.

Equivalence relations, which guarantee that the models will have the same behaviors, usually do not decrease the number of states sufficiently. Instead, simulation relations, which relate models to their abstractions, are used. The simulation guarantees that every behavior of a model is a behavior of its abstraction. However, the abstraction might have behaviors that are not possible in the original system.

Abstract state spaces may be obtained by means of under-approximation methods, which remove behaviors, or over-approximation methods, which add new behaviors. Thus, in case of under-approximation, a bug in the abstract model implies a bug in the concrete model, and in case of over-approximation, correctness of the abstract model implies correctness of the concrete model. Further in this article we only consider over-approximations, also known as conservative abstractions.

Developing abstract models involves finding a compromise between two conflicting goals: 1) generation of small abstract models that can be model checked; 2) generation of precise abstract models.

Usually, the smaller the model, the more behaviors it allows. This may lead to spurious counterexamples that are not present in the concrete model. There are at least two ways out: 1) construction of precise abstract models; 2) analysis of counterexamples and modification of the abstract model according to the acquired information (counterexample-guided abstraction refinement).

Methods that create precise abstract models (for example, based on counter abstraction or environment abstraction [4]) lead to models of big size in case of complicated protocols.

The idea of compositional verification [5] is to exploit the natural decomposition of a distributed system into processes. Processes are verified individually (with a generalized environment), then the results are combined, and a verdict about correctness of the initial model is made. A compositional approach must provably lead to simplified models satisfying the properties of the initial model.

V. A METHOD OF COMPOSITIONAL MODEL CHECKING

A. General Idea

The method described in this paper adapts the method [6] to work with a subset of Promela. The method is based on a combination of model checking and theorem proving. The choice of Spin is motivated by the fact that Spin is a modern and constantly evolving tool that supports many optimizations and verification modes. The Promela language is convenient for description of distributed systems, including cache coherence protocols. Moreover, Spin may be used as the basis for generators of test programs the purpose of which is

verification of implementations of cache coherence protocols [7].

The method shows how to build an abstract model that simulates a given concrete model of a cache coherence protocol. The construction is performed by means of syntactic transformations of the concrete Promela model.

B. A Mathematical Model of Cache Coherence Protocols

Cache coherence protocols may be seen as asynchronous systems of communicating processes in which a process is a finite automaton. Then a mathematical model of a cache coherence protocol is a system of communicating finite automata.

A Promela model specifies the behavior of a set of asynchronously executing processes in a distributed system. Each Promela process defines an extended finite automaton. Thus, Promela is suitable for describing models of cache coherence protocols.

By simulating the execution of a Promela model we can build a digraph of all reachable states of the model. Each node in the graph represents a state of the model, and each edge represents a single possible execution step by one of the processes. This graph is always finite [8].

Safety properties can be interpreted as statements about the presence or absence of specific types of nodes in the reachability graph.

Let us consider the transition system corresponding to the reachability graph. The following discussion considers a subset of Promela.

A transition system is a triple $TS = (S, S_0, E)$, where S is a finite non-empty set of states, $S_0 \subseteq S$ is a non-empty set of initial states, $E \subseteq S \times S$ is a transition relation on S such that

$$(\forall s \in S) (\exists s' \in S): (s, s') \in E$$

In order to be able to formally define syntactic transformations of a Promela model, we will represent models by means of a triple $P = (V, \Theta, R)$, where

- V is a set of variables of the model, each variable is of its own type;
- Θ is the initialization predicate;
- R is the set of transition rules represented as guarded commands consisting of a condition and a set of assignments:

$$cond \rightarrow \{v_1 := t_1; \dots; v_k := t_k\},$$

where $cond$ is the condition (predicate), $v_i \in V$ are model variables, each t_i is a term of the same type as v_i ; $:=$ denotes assignment.

An interpretation of a set of typed variables V is a mapping that assigns to each variable $v_i \in V$ a value in the domain of v_i .

A triple $P = (V, \Theta, R)$ determines a transition system $TS^P = (S, S_0, E)$ in the following way. Each state $s \in S$ is an interpretation of the set V . For every term t we write $s(t)$ for the value of t in the state s . For a predicate φ , we denote $s \models \varphi$ if and only if $s(\varphi) = true$. A predicate φ is an *invariant* of a model P , denoted by $P \models \varphi$, if $\forall s \in S: s \models \varphi$. S_0 is the set of states $s \in S$ such that $s \models \Theta$.

There exists a transition $s \rightarrow s'$, which means $(s, s') \in E$, if there exists a transition rule

$$cond \rightarrow \{v_1 := t_1; \dots; v_k := t_k\},$$

such that $s \models cond$ and s' is a state in which

$$(\forall i \in \{1, \dots, k\}) (s'(v_i) = s(t_i))$$

and

$$(\forall v_j \in V \setminus \{v_1, \dots, v_k\}) (s'(v_j) = s(v_j)).$$

C. The Abstract Model

Let $N = \{p_1, \dots, p_n\}$ be a parameter set, where p_1, \dots, p_n are constants of the type used to represent processes in the model and n is a natural number defined by the number of cache agents in the system.

Let $P = (V, \Theta, R)$ be a symmetric model [9] and $M = \{p_1, \dots, p_m\}$ be a subset of the set $N = \{p_1, \dots, p_n\}$, $m \leq n$. Let abs be the element that is an abstraction of elements p_{m+1}, \dots, p_n and $M_{abs} = M \cup \{abs\}$. We define the abstract model $P_{abs} = (V, \Theta_{abs}, R_{abs})$ with the parameter set M_{abs} as follows.

Let S be the set of states of the model P and S_{abs} be the set of states of the model P_{abs} .

The predicate Θ_{abs} is obtained by the syntactic transformations $Trans_P$.

The transition rules R_{abs} are obtained by syntactic transformations $Trans_R$ that include transformations of conditions $Trans_P$ and transformations $Trans_A$ of the assignments that appear in the rules:

$$\begin{aligned} Trans_R(cond \rightarrow \{v_1 := t_1; \dots; v_k := t_k\}) = \\ Trans_P(cond) \rightarrow \{Trans_A(v_1 := t_1); \dots; Trans_A(v_k := t_k)\} \end{aligned}$$

The transformations of terms $Trans_T$ are defined in the following way.

$$Trans_T(v) = v \text{ for each } v \in V,$$

$$Trans_T(p_i) = \begin{cases} p_i & \text{for } i \leq m, \\ abs & \text{for } i > m \end{cases},$$

$$Trans_T(c) = c \text{ for all other constants } c.$$

This definition is extended inductively to work with composite term expressions.

Suppose $\varphi(t_1, \dots, t_k)$ is a predicate, i.e., a logical combination of t_1, \dots, t_k . Then $Trans_T(\varphi(t_1, \dots, t_k))$ is the same logical combination of $Trans_T(t_1), \dots, Trans_T(t_k)$. Define $Trans_P(\varphi)$ to be the same logical combination of t'_1, \dots, t'_k , where

$$t'_i = \begin{cases} t_i, & \text{if } Trans_T(t_i) = t_i, \\ true, & \text{if } Trans_T(t_i) \neq t_i \text{ and } t_i \text{ occurs positively in } \varphi, \\ false, & \text{if } Trans_T(t_i) \neq t_i \text{ and } t_i \text{ occurs negatively in } \varphi. \end{cases}$$

Now let us define the transformations of assignments $Trans_A$. Denote by \emptyset the absence of assignment and let

$$t' = \begin{cases} t, & \text{if } Trans_T(t) = t, \\ \text{any value in the domain of } t, & \text{otherwise} \end{cases}.$$

Table 1 lists the allowed types of assignments and their corresponding transformations. Define *Array* to be a Promela array and $f_2 : N \rightarrow M_{abs}$ to be a mapping that maps p_1, \dots, p_m to themselves and maps p_{m+1}, \dots, p_n to *abs*.

The abstract set of transitions is defined as follows:

$$R_{abs} = \{Trans_R(r) \mid r \in R\}.$$

D. Justification of the Abstraction Rules

It can be shown [9] that the abstraction map $\alpha : S \rightarrow S_{abs}$ preserves transitions, that is

$$\forall s \in S : (s \rightarrow s') \Rightarrow (\alpha(s) \rightarrow \alpha(s'))$$

Then, safety properties are preserved: If a state is reachable in the concrete model, it is reachable in the abstract model. In other words, the abstraction map is a simulation relation.

Table 1. Syntactic Transformations of Assignments

Type of assignment	Assignment transformation
$v := t$	$v := t'$
$Array[p_i] := t$	\emptyset , if $i > m$ $Array[p_i] := t'$, if $i \leq m$
$Array[t] := p_i$	$Array[t] := f_2(p_i)$

E. The Method

The verification method is based on two observations. The first one is the fact that the abstraction map is a simulation relation. The second one is the guard strengthening principle [9] that makes the following strategy correct.

Given a model P and a predicate φ , in order to prove that $P \models \varphi$: 1) add φ to the conditions of transition rules of P by means of conjunction; 2) prove that φ is an invariant of the newly acquired model.

The method consists of the following steps. Input objects are a symmetric model P with parameter set $N = \{p_1, \dots, p_n\}$ and a safety property φ .

1. Construct P_{abs} , using the syntactic transformations from section V.C. Let $Q = P_{abs}$.
2. If $Q \models \varphi$, the verification is finished: we conclude that $P \models \varphi$.
3. Otherwise, examine a counterexample provided by Spin, devise an invariant ψ and modify Q as described in [9]. Set $\varphi = \varphi \wedge \psi$. Go to step 2.

VI. DESIGN OF A CACHE COHERENCE PROTOCOLS VERIFICATION SYSTEM

The syntactic transformations described in section V.C can be fully automated. Performing them by hand is tedious and impractical, especially in an industrial setting. Therefore, in order to alleviate this problem, a tool may be developed, which would build an internal representation of the concrete Promela model, modify it according to the transformations, and produce the abstract model. An abstract syntax tree may be the internal representation.

The transformations of Promela models are shown in Fig. 1.

The question of automating the refinement transformations is significantly harder. Further research is needed in this direction.

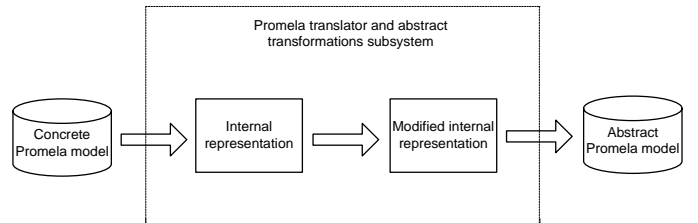


Figure 1. The transformations of Promela models

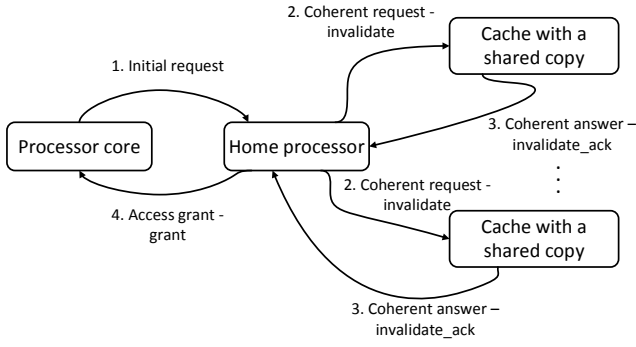


Figure 2. Processing of the read/write requests of the German cache coherence protocols

VII. VERIFICATION OF THE GERMAN CACHE COHERENCE PROTOCOL

I developed a Promela model of the German protocol. The model is written in the style of [10]. The model implements the algorithm of memory access requests processing shown in Fig. 2.

A processor core and the corresponding cache controller are represented by the Promela process `core` and the home-processor is represented by the process `home`. Thus, the model consists of one process `home` and N processes `core` where N is a natural number. Interaction between the processes is accomplished by means of the three Promela arrays `channel1`, `channel2`, and `channel3` (see Fig. 3).

The array `channel1` is for the initial requests `req_*` sent by a processor to the home processor. The array `channel2` is for the snoop requests `invalidate` sent by the home processor to cache controllers and for grants `grant_*`. The array `channel3` is used for coherence answers sent by cache controllers to the home processor (`invalidate_ack`).

The German protocol uses three main states of a cache line: Invalid, Exclusive, and Shared.

According to the transformations described in section V.C, I developed the initial version of the abstract model. The abstract model contains one process `home`, two processes `core`, and one abstract process `home_abs`. One of the most complicated parts of creating the abstract model – the transformation of assignments – is depicted in Table 2. Table 2 shows examples of the corresponding transformations of the German cache coherence protocol Promela model.



Figure 3. Communication channels between processes in the Promela model of the German cache coherence protocol

Table 2. Examples of the syntactic transformations of the Promela model of the German protocol

Assignment	Assignment transformation
<code>curr_command = req_shared</code>	<code>curr_command = req_shared</code>
<code>sharer_list[i] = true</code>	\emptyset , if $i > m$ <code>sharer_list[i] = true</code> , if $i \leq m$
<code>curr_client = i</code>	<code>curr_client = i</code> in a concrete process <code>curr_client = abs</code> in the abstract process

The verified property stated that it is impossible for a cache line to be in state Exclusive in one cache and in state Shared in some other cache. For example:

```
never { do :: assert( !(cache[0] == exclusive && cache[1] == shared)) ) od }
```

This property did not hold on the initial abstract model. According to section V.E, I performed the refinement process. Two additional invariants were developed and the verification process was finished due to the absence of counterexamples. The refinement process was similar to that described in [6].

For the experimental check of the method's ability to find bugs, I verified two buggy versions of German described in [4]. In the first buggy version, after the home processor grants exclusive access to a cache, it fails to set the `exclusive_granted` variable to true. Thus, when another cache requests shared access, it gets the access even though the first cache holds it in exclusive state. In this case Spin issues a counterexample because the assertion

```
assert( !(cache[0] == exclusive && cache[1] == shared)) )
```

is violated.

In the second buggy version, the home processor grants a shared request even if `exclusive_granted` variable is true. In this case Spin issued a counterexample because of the violation of one of the invariants found during the abstraction process.

VIII. CONCLUSION AND DIRECTIONS FOR FUTURE WORK

Formal methods for verification of cache coherence protocols fall into two groups: methods based on model checking and methods based on deductive verification. Model checking is fully automated but suffers from the state space explosion problem. Deductive verification is scalable but requires a lot of expert's hand work. Combination of the two approaches seems promising because of its potential ability to lead to a scalable method that requires an acceptable amount of hand work.

On the basis of existing literature, a method that is such a combination is described. Although the method can be used for parameterized verification, it has some drawbacks. It

supports a very limited subset of Promela constructs and poses unnecessary limitations on the way verification engineers should write their Promela models. The style of the Promela model used in this paper is less intuitive than the style of the model described in [7]. The model from [7] was obtained by a natural decomposition of the Elbrus system-on-chip under verification and organizing process communication through Promela channels. The model was successfully used in verification of several Elbrus systems.

Future work directions include provable extension of the Promela subset that can be dealt with by the verification method, the examination of the impacts of different styles of descriptions of cache coherence protocols, and development of tools that would automate parts of the verification process. The verification process will be applied to Elbrus microprocessors.

REFERENCES

- [1] Z. Manna, A. Pnueli, "The temporal logic of reactive and concurrent systems: specification," Springer-Verlag, 427 pp., 1992.
- [2] E.M. Clarke, O. Grumberg, D. Peled, "Model checking," MIT Press, 314 pp., 1999.
- [3] S. Park, D. Dill, "Verification of FLASH cache coherence protocol by aggregation of distributed transactions," Proceedings of the 8th annual ACM symposium on parallel algorithms and architectures, pp. 288–296, 1996.
- [4] M. Talupur, "Abstraction Techniques for Parameterized Verification," PhD Thesis, 2006.
- [5] E. Clarke, D. Long, K. McMillan, "Compositional model checking," Proceedings of the fourth IEEE symposium on logic in computer science, 1989.
- [6] C. Chou, P. Mannava, S. Park, "A simple method for parameterized verification of cache coherence protocols," Formal methods in computer-aided design, vol. 3312, pp. 382–398, 2004.
- [7] V. Burenkov, "Generator testov dlya verifikatsii protokola cogerentnosti kesh pamyati [A test generator for cache coherence protocol verification]," Voprosi radioelektroniki, seria EVT, 3, pp. 56–63, 2014.
- [8] G. Holzmann, "The Spin model checker: primer and reference manual," Addison-Wesley Professional, 608 pp., 2003.
- [9] S. Krstic, "Parameterized system verification with guard strengthening and parameter abstraction," Automated verification of infinite state systems, 2005.
- [10] A. Pnueli, S. Ruah, L. Zuck, "Automatic deductive verification with invisible invariants," Tools and algorithms for the construction and analysis of systems, vol. 2031, pp. 82–97, 2001.

A Model-Based Approach to Design Test Oracles for Memory Subsystems of Multicore Multiprocessors

Alexander Kamkin
ISP RAS
Moscow, Russian Federation
kamkin@ispras.ru

Mikhail Petrochenkov
MCST
Moscow, Russian Federation
petroch_m@mcst.ru

Abstract—The paper describes a method for constructing test oracles for memory subsystems of multicore microprocessors. The method is based on using nondeterministic reference models of systems under test. The key idea of the approach is on-the-fly determinization of the model behavior by using reactions from the system. Every time a nondeterministic choice appears in the reference model, additional model instances are created and launched (each simulating a possible variant of the system behavior). When the testbench receives a reaction from the system under test, it terminates all model instances whose behavior is inconsistent with that reaction. An error is detected if there is no active instance of the model. The suggested method was used to verify the L3 cache of Elbrus-8C microprocessor and allowed finding three bugs.

Keywords—multicore microprocessors, cache memory, memory consistency, coherence protocols, functional verification, model-based testing, testbench automation, test oracle, Elbrus-8C.

I. INTRODUCTION

A key feature of modern microprocessor architectures is *multicoreness*, which is implementation of several processing units, so-called *cores*, on a single chip. To reduce time to access data from the main memory, each core has a local cache, often with two levels, L1 and L2; in addition, all cores can share the L3 cache. The presence of several data storages makes it possible to have multiple copies of the same data within the system and requires special mechanisms to ensure the storages to be in a *coherent state*. At the heart of such mechanisms is a *coherence protocol*, a set of rules that governs interactions between storage devices and guarantees memory consistency for all possible data access scenarios [1].

State-of-the-art coherence protocols are complicated; their implementation in hardware is difficult and error-prone. Accordingly, thorough verification of memory subsystems is required [2]. A widely accepted approach to ensure the correctness of complex hardware designs is *simulation-based verification*, or *testing*. A *test system*, also known as a *testbench*, solves two main tasks: first, it generates a stream of stimuli; second, it checks whether the design behavior satisfies the requirements [3]. This paper addresses the second problem, i.e. checking the reactions of a memory subsystem in response to an arbitrary series of stimuli; it introduces a method for constructing *test oracles* (reaction checkers) based on high-level reference models of memory subsystems.

The rest of the paper is organized as follows. Section II reviews the existing techniques for designing test oracles. Section III suggests an approach to the problem. Section IV describes a case study on using the suggested approach in an industrial setting. Section V concludes the paper.

II. RELATED WORK

A memory subsystem as an object of testing has a number of distinctive features that should be taken into consideration when designing a test oracle. First, it consists of many devices that work in parallel and can receive requests (*stimuli*) and send responses (*reactions*) through several input and output channels (interfaces with the microprocessor cores). Second, its behavior essentially depends on the order of requests to separate data blocks (*cache lines*); which, in turn, depends on the time of the requests initiation as well as on the subsystem's microarchitecture. Third, requests to a single cache line are processed mostly one at a time (in other words, requests are *serialized*).

It is also worth considering how reference models of memory subsystems are developed. Many implementation details, such as the timing of request execution, are typically ignored: operations are described as atomic actions, while interactions between blocks are modeled by “zero-time” function calls. Such kind of models are often called *functional models*. The simplified nature of reference models makes them more tolerant to changes in the subsystem implementation, but at the same time it makes building test oracles a more difficult task. Models of that kind cannot predict the exact order of request execution basing solely on the request timestamps. In this sense, functional models are surely *nondeterministic*. The problem of building test oracles from nondeterministic models is well known; there are several approaches to solve it.

In [4], a reference model (*specification*) and a system under test (*implementation*) are represented as *Partial Order Input/Output Automata*. In such an automaton, each transition is labeled not by a “stimulus-reaction” pair, but by a *partially ordered multiset* (multiple stimuli and reactions are allowed). An implementation is said to *conform* to its specification if for each specification trace there is an implementation trace of the same length in which the order of events corresponds to the order given in the specification trace. A similar approach is presented in [5], where a model of *Asynchronous Finite State Machine* is used. In both methods, checking is carried out

some time after the last stimulus (the time should be long enough to allow all reactions to occur and the implementation to enter in a stationary state). The scheme is applied under the assumption that a stimulus generator is “idle” every now and then during testing.

In [6], a similar concept of correspondence is used, but the approach focuses on “continuous” event flows (with no stops in stationary states). A test oracle is based on a so-called *trace matcher*, which acts as follows: it receives reactions from the specification and the implementation and adds them into the corresponding partially ordered multisets (Y is for the specification, and Z is for the implementation); before adding reactions, the minimal (in a sense of the precedence relation) events ($\min(Y) \cap \min(Z)$) are removed from both multisets; if the amount of time a reaction stays in a multiset exceeds some predefined limit, an error is indicated. As compared with [4] and [5], the method requires more deterministic reference models: the order of implementation reactions may not be the same as of specification ones, but the sets of specification and implementation reactions should coincide (this requirement can be weakened by marking some reactions as being *optional*). To apply the approach to a complex system, a testbench needs to use “hints” from the implementation that help to decide what functionality of the reference model is to be executed [7].

Our work tries to combine [4] and [6]: it allows using nondeterministic models without restrictions on test sequences and without using “hints” from implementations. A general approach is as follows. As soon as there are several possible ways to continue the execution of the reference model (such a situation is referred to as a *nondeterministic choice*), additional instances of the model are created and launched (the base instance goes on with one of the branches). When the testbench receives a reaction from the device under test, the reaction itself and its characteristics (such as a response type, message data, etc.) are used to determine what behavior is infeasible and what instances to terminate. If there is no active instance of the reference model, an error is reported. Obviously, in the general case the number of states (and variants of behavior) grows exponentially with the number of decision points. However, for memory subsystems the suggested scheme can be effectively implemented: first, requests to different cache lines are almost independent (existing dependencies can be neglected); second, requests to a single cache line are serialized.

III. SUGGESTED APPROACH

Let us clarify what kind of reference models are used by test oracles for checking behavior of memory subsystems. Stimuli are divided into two groups: *primary stimuli*, which are requests from clients (cores, controllers, etc.) to perform certain operations with the memory, and *secondary stimuli*, which are responses of the test environment to some reactions of the memory subsystem (every reaction and every secondary stimulus is caused by some primary stimulus). A *memory subsystem model* is decomposed into a number of *operation models*, one for each type of primary stimulus. An operation model has the following interface (the detailed structure is not of importance):

- $p \leftarrow \text{start}(x)$ – the model creates a process p that handles the primary stimulus x ;
- $p.\text{receive}(x)$ – the process p receives the secondary stimulus x from the environment;
- $p.\text{send}(y)$ – the process p sends the reaction y to the environment (a callback function);
- $p.\text{finished}()$ – the model checks whether the process p has completed.

From the structural point of view, a memory subsystem model consists of *cache line models* and a *switch*. Given a stimulus, the switch determines what cache line is addressed and sends the stimulus to the corresponding model. A cache line model works as follows. To preserve the order of requests from the same client, it has a set of *request queues*, Q_1, \dots, Q_N , where N is a number of clients (only requests from the heads of the queues can be processed). Additionally, it contains a *state model*, which represents data stored in the cache line and auxiliary information that affects the behavior of the operation models. A cache line model is nondeterministic and can be described by the following pseudo-code:

```

while true do
  wait  $\bigvee_{i=1,N} (Q_i \neq \emptyset)$ 
   $Q \leftarrow \{(\text{head}(Q_i), i) \mid i \in \{1, \dots, N\} \wedge (Q_i \neq \emptyset)\}$ 
   $(x, i) \leftarrow \text{select}(Q)$ 
   $\text{dequeue}(Q_i)$ 
   $p_i \leftarrow \text{start}(x)$ 
  wait  $p_i.\text{finished}()$ 
end

```

If there are requests from clients ($\bigvee_{i=1,N} (Q_i \neq \emptyset)$), a set of candidates for processing (Q) is built. After that, one of the requests is nondeterministically selected ($(x, i) \leftarrow \text{select}(Q)$). The chosen request is removed from the corresponding queue ($\text{dequeue}(Q_i)$), and its processing is initiated ($p_i \leftarrow \text{start}(x)$). When the process is completed ($p_i.\text{finished}()$), the procedure described above is repeated.

A cache line model has the following interface methods:

- $\text{receive}(x, i) \equiv \text{enqueue}(Q_i, x)$ – the model receives the primary stimulus x from the client i ;
- $\text{receive}(x) \equiv p.\text{receive}(x)$ – the model receives the secondary stimulus x from the environment.

The test oracle structure follows from the reference model structure: one can distinguish a *memory subsystem oracle*, a *cache line oracle* and an *operation oracle*. An oracle of each type is built upon a model of the corresponding type. Thus, a memory subsystem oracle consists of cache line oracles and a switch; a cache line oracle includes request queues, operation oracles, a state model and a *message matcher* (the component’s functions will be described later on); an operation oracle contains an operation model. It should be noted that there is a distinction between oracle and model switches: an oracle switch routes not only stimuli but also reactions. The design of a cache line oracle based on operation oracles is of the most interest (see Fig. 1).

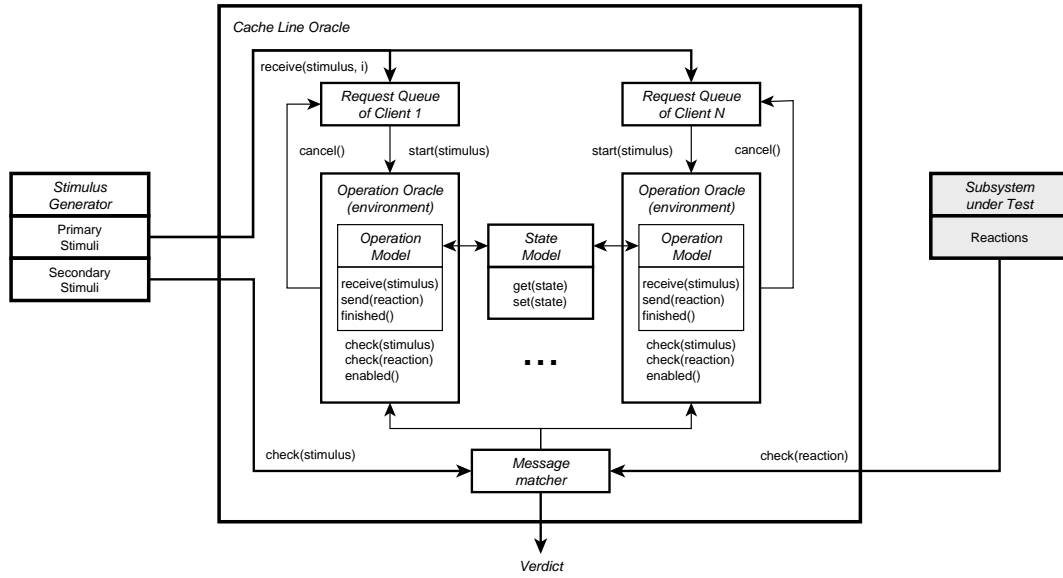


Figure 1. Structure of a cache line oracle

An operation oracle checks the correctness of reactions (and possibly validity of secondary stimuli) for the individual operation (provided that this operation is processed by the memory subsystem). A cache line oracle does not impose any restrictions on how operation oracles are implemented. If a set of reactions caused by the operation depends solely on the cache line state, the approach presented in [6] can be applied. In the simplest case, checking is carried out as follows. Every time the operation model invokes *send(y)*, the reaction *y* is added to the multiset *Y*. When receiving a reaction *z* from the implementation, the *check(z)* method of the operation oracle is called. It checks whether *z* belongs to *Y*: in case of the positive answer, *z* is removed from *Y*; otherwise, the error is indicated. Also, the operation oracle overrides the *finished()* method of the operation model: in addition to checking the operation completion, it tests whether the set *Y* is empty.

The model does not provide enough information to determine the exact order in which requests from different clients are handled. A cache line oracle launches the operation oracles for all possible request choices in parallel (only one request is to be processed by the memory subsystem, but for now, one cannot decide which one). The cache line oracle is described by the following pseudo-code (p_i refers to an operation oracle for the client *i*):

```

while true do
  wait  $\vee_{i=1..N} enabled(Q_i)$ 
   $Q \leftarrow \{(head(Q_i), i) \mid i \in \{1, \dots, N\} \wedge enabled(Q_i)\}$ 
  for  $(x, i) \in Q$  do
    dequeue( $Q_i$ )
     $p_i \leftarrow start(x)$ 
  end
end
enabled( $Q_i$ )  $\equiv (Q_i \neq \emptyset) \wedge ((p_i = null) \vee p_i.finished())$ 

```

The message matcher analyzes implementation reactions (and possibly secondary stimuli) and identifies the request being executed by the memory subsystem. Having received a reaction *z* from the implementation, the *check(z)* method of the message matcher is invoked, which, in turn, calls *check(z)* in all active $((p_i \neq null) \wedge \neg p_i.finished())$ operation oracles.

```

count  $\leftarrow 0$ 
for  $i \in \{1, \dots, N\}$  do
  if  $(p_i \neq null) \wedge \neg p_i.finished()$  then
    if  $p_i.check(z)$  then
      count  $\leftarrow count + 1$ 
    else
       $p_i.cancel()$ 
       $p_i \leftarrow null$ 
      push( $Q_i, x$ )
    end
  end
end
assert (count  $\neq 0$ )

```

If an operation oracle (p_i) returns the negative verdict ($p_i.check(z) = false$), the oracle process is forcibly stopped ($p_i.cancel()$), and the primary stimulus having initiated the process is returned to the head of the corresponding queue ($push(Q_i, x)$). If there are no active processes ($count = 0$), then the cache line oracle returns the negative verdict. Secondary stimuli are handled in a similar way; a difference is that if an operation oracle's verdict is positive ($p_i.check(x) = true$), the stimulus is transmitted to the operation model ($p_i.receive(x)$).

To construct a test oracle in the suggested way, a system under test is expected to meet the following conditions (in addition to request serialization): first, the behavior of each operation is unambiguously defined by the system state at the operation start time; second, each operation changes the global state of the system just before its completion; third, a client

being served can be unambiguously identified by matching primary requests with reactions.

IV. CASE STUDY

The presented method for designing test oracles was used to develop a test system for the L3 cache of the Elbrus-8C octal-core microprocessor (total volume – 16 MB; size of a cache line – 64 B; number of banks – 8; bank associativity – 16) [8]. The L3 cache is a point of serialization for the *read* and *write* requests from the microprocessor cores and the *snoop* requests (auxiliary requests for maintaining cache coherence) from the system interface controller. For each message it is possible to identify the affected cache line; for this purpose, the oracle switch stores a relation between primary request addresses and resource identifiers used in reactions and secondary stimuli. In general, the cache line oracle follows from the suggested scheme, but has some particular features described below.

First of all, operations on cache lines of the same set (cache lines located at the same index) are surely dependent: inclusion of a cache line might trigger eviction of another one. It should be emphasized that a victim line cannot be determined without using a cycle-accurate reference model and without getting “hints” from the implementation. To solve this problem and to make all cache lines to be served independently, we assume that any cache line (whose state is not *Invalid*) can be evicted at any moment. This assumption is implemented by adding a virtual client *Eviction* to all cache line oracles (such a trick is legal, because eviction requests are serialized like any other stimuli).

In most of the cases, a requesting client can be identified based on reactions, but there are two exceptions. First, *writing data with eviction from L2 (Write-Back)* – if the data are not in the L2 cache, the request is canceled (it completes without sending any reaction and without changing the state). Second, *prefetching data into L3 (Prefetch)* – if the data are in the L3 cache, the request is canceled. The first situation is solved by forcibly stopping a model of the *Write-Back* operation as soon as it is known that the core (the L2 cache of the core) has no data (such a solution is correct, because requests from cores cannot load data into other cores; requests from the requesting core cannot be chosen until the *Write-Back* operation is completed). The second problem is solved by “detaching” the prefetch requests from the cores and moving them to additional clients (the completion of a prefetch request is detected indirectly by identifying the completion of one of the following requests from the same core).

If a cache line (stored in the L3 cache) is in the *Shared* state and no core has its copy in the L2 cache, the line can be evicted (become *Invalid*) without sending messages to the environment. Therefore, if a cache line model is in the *Shared* state, it means that the corresponding cache line of the implementation is either *Shared* or *Invalid*. Being executed in the *Shared* state (without copies of the data in the cores), an operation oracle spawns two operation models: one operates on the assumption that the line is *Shared*; the other operates on the assumption that the line is *Invalid*.

It should be noted that L3 under test has no strict requirements on serialization of so-called *special operations* (noncoherent reads and uncacheable writes). It is allowed to concurrently process any number of such operations over the same cache line. This exception does not complicate the test oracle structure: first, special requests are permitted only in the *Invalid* state (otherwise, an eviction starts); second, special operations do not change the state of the cache and do not affect other operations.

The use of the suggested approach allowed to discover three errors in the L3 design. The first one concerns the operation of *reading data with storing them in L3 (R32L3 and R64L3)* – the internal directory erroneously marks the line as having been stored in the L2 cache of the requesting core. The second one is an unnecessary delay in data eviction caused by a special operation. Finally, the third one relates to reading of invalid data from the write-back buffer.

V. CONCLUSION

Memory subsystems of multicore microprocessors are extremely complex devices; their implementation should be thoroughly tested. Test oracles play a key role in testbench automation; the main part of an oracle is a reference model, i.e. a simplified software implementation of the device under test. Models of memory subsystems are usually nondeterministic in a sense that given a set of stimuli, one cannot accurately determine a set of reactions. In this article, we have proposed a method for designing test oracles for memory subsystems based on reaction-driven refinement of a set of behavior variants. An error is reported if the refinement process leads to the empty set of variants. The suggested approach was applied to verify the L3 cache of Elbrus-8C microprocessor and allowed finding three errors.

REFERENCES

- [1] Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.
- [2] Kamkin A., Petrochenkov M. Sistema podderzhki verifikatsii realizatsii protokolov kogerentnosti s ispol'zovaniem formal'nykh metodov [A system to support formal methods-based verification of coherence protocol implementations]. Voprosy radioelektroniki, seriya EVT, 2014, 3. p. 27-38.
- [3] Bergeron J. Writing Testbenches: Functional Verification of HDL Models. Kluwer Academic Publishers, 2000. 354 p.
- [4] von Bochmann G., Haar S., Jard C., Jourdan G.V. Testing Systems Specified as Partial Order Input/Output Automata. ICTSS, 2008. p. 169-183.
- [5] Kuliain V., Petrenko A., Pakoulin N., Kossatchev A., Bourdonov I. Integration of Functional and Timed Testing of Real-Time and Concurrent Systems. PSI, 2003. p. 450-461.
- [6] Chupilko M., Kamkin A. Runtime Verification Based on Executable Models: On-the-Fly Matching of Timed Traces. MBT, EPTCS 111, 2013, p. 67-81.
- [7] Baratov R., Kamkin A., Maiorova V., Meshkov A., Sortov A., Yakusheva M. Trudnosti modul'noi verifikatsii apparatury na primere bufera komand mikroprotessora «El'brus-2S» [Difficulties of the unit-level hardware verification on the example of the instruction buffer of the Elbrus-2S microprocessor]. Voprosy radioelektroniki, seriya EVT, 2013, 3. p. 84-96.
- [8] Kozhin A., Kozhin E., Kostenko V., Lavrov A. Kesh tret'ego urovnya i podderzhka kogerentnosti mikroprotessora «El'brus-4S+» [L3 cache and cache coherence support in «Elbrus-4C+» microprocessor]. Voprosy radioelektroniki, seriya EVT, 2013, 3. p. 26-38.

An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms

Alexander Kamkin, Alexander Protsenko, Andrei Tatarnikov
Institute for System Programming of the Russian Academy of Sciences (ISP RAS)
Moscow, Russian Federation
Email: {kamkin, protsenko, andrewt}@ispras.ru

Abstract—In this work, an approach to generate test programs for functional verification of memory management units of microprocessors is proposed. The approach is based on formal specification of memory access instructions, namely load and store instructions, and memory devices such as cache units and address translation buffers. The use of formal specifications helps automate development of test program generators and makes verification systematic due to clear definition of testing goals. In the suggested approach, test programs are constructed by using combinatorial techniques, which means that stimuli – sequences of loads and stores – are created by enumerating all feasible combinations of instructions, situations (instruction execution paths) and dependencies (sets of conflicts between instructions). It is of importance that test situations and dependencies are automatically extracted from specifications. The approach has been used in a number of industrial projects and allowed to discover critical bugs in memory management mechanisms.

Keywords—microprocessors, memory management, caching, address translation, functional verification, formal specifications, test program generation, instruction stream generation.

I. INTRODUCTION

A computer memory is known to be a complex hierarchy of data storage devices varying in volume, latency and price [1]. In addition to registers and main memory, microprocessors include a multi-level cache memory and address translation buffers. The set of devices responsible for handling memory accesses is referred to as a *memory subsystem* or a *memory management unit (MMU)*. Being one of the key microprocessor components, the memory subsystem is strongly required to be correct and reliable. Due to the complicated structure of the memory, the number of situations that can occur in processing load and store instructions is huge; this makes it improbable to verify the subsystem “manually”.

In the current practice, tests – programs in the assembly language of the microprocessor under test – are created in an automated way with the intensive use of random generation. A tool that constructs test programs is called a *test program generator (TPG)* or an *instruction stream generator (ISG)* [2]. In a typical use case, a TPG accepts probability distributions for instructions types and operand values as well as other parameters and produces a set of programs in compliance with the settings. Though the randomization-based approach is able

to find “high-quality” bugs, it is not systematic and does not guarantee the verification completeness.

In the present work, an approach to generate test program for memory subsystems of single-core microprocessors is discussed (the multi-core issues, such as memory consistency and cache coherence [3], are out of the scope of the paper). The proposed approach complements the random-based testing and enables thoroughly checking situations in the MMU behavior. It uses specifications of memory access instructions, i.e. load and store instructions, and specifications of memory devices including, first of all, caches and address translation buffers. The formal specifications serve as a source of test coverage information and allow automatically extracting instruction-level situations and dependencies. Test programs are built by composing possible situations and dependencies for instruction sequences of bounded length.

The rest of the paper is organized as follows. Section II is a primer on microprocessor memory organization. Section III provides a brief overview of the related work. Section IV describes in detail the mentioned approach to test program generation. Section V considers industrial applications of the described approach. Finally, Section VI concludes the paper and outlines directions for future research and development.

II. MEMORY SUBSYSTEM

In a nutshell, a memory subsystem of a microprocessor is intended for handling memory accesses, namely instruction fetch requests, data loads and data stores. Its functions include translation of virtual addresses into physical ones, memory protection, code and data caching, etc. [1]. Let us consider the essential concepts of the memory management.

From a programmer’s perspective, a computer memory is a linear array of bytes. However, the underlying mechanisms and techniques – usually referred to as a *virtual memory* – are rather sophisticated. A *virtual address space*, i.e. a range of the byte array indices available for programs to use, is commonly divided into disjoint *segments*. Given a segment and a virtual address, the MMU acts as follows. If the microprocessor mode satisfies the segment’s privilege level, the virtual address is translated into the *physical address*, and an access to the *physical memory* is performed; otherwise, an address error exception is thrown.

Segments are divided into *mapped* and *unmapped*; the latter, in turn, are subdivided into *cached* and *uncached*. Addresses of mapped segments are translated with the help of *translation lookaside buffers (TLB)*, which store the mapping between *virtual page numbers (VPN)* and *physical frame numbers (PFN)*. If there is a match, the VPN bits of the virtual address are replaced with the PFN bits, and the process continues. Otherwise, a TLB refill exception is thrown, which triggers the operating system to look up the page table and update the TLB. Unmapped addresses are translated directly with no use of the buffers. Accessing cached segments, as opposed to uncached ones, activates the caching mechanisms.

A cache is an intermediate storage responsible for speeding up access to frequently used data. An average microprocessor has two- or three-level cache memory. Typically, an L_i cache stores a subset of L_{i+1} contents; the highest-level cache is the largest one; it interacts immediately with the main memory. A cache works as follows. As soon as data are requested, the cache controller checks whether they are in the buffer. If they are (it is said to be a *cache hit*), the data are taken from there and returned to the requester. Otherwise (it is said to be a *cache miss*), the controller chooses a victim among the data blocks stored in the buffer and replaces it with the data loaded from the higher-level cache or the main memory.

In the general case, a cache comprises a number of *sets*; each set consists of a number of *lines*; each line includes *data* and a *tag*. Let $S = 2^s$ be the number of sets; W be the number of lines in a set; $B = 2^b$ be the size of a data block. Depending on the values of S and W , the following types of cache memory are recognized: (1) a *direct-mapped cache* ($W = 1$); (2) a *fully associative cache* ($S = 1$); (3) a *set-associative cache* ($W > 1$ and $S > 1$). The bit representation of an address is interpreted as follows: the bits $[0, \dots, b-1]$ refer to a byte inside a data block; $[b, \dots, b+s-1]$ identify a set; $[b+s, \dots, m-1]$, where m is the address length, define a tag. To determine whether the cache contains data for a given address, first, the set is identified; then, the tags of the set's lines are concurrently compared with the tag extracted from the address. If there is a match, then the requested data are available in the cache.

III. RELATED WORK

There are several TPG tools based on formal specifications of memory subsystems. DeepTrans (IBM Research) [4] is one of them. The approach is targeted at testing address translation mechanisms and uses a special-purpose modeling language. A process of address translation is depicted as a directed acyclic graph whose vertices correspond to the process stages and whose edges relate to the transitions between the stages. A path from the source of the graph to the sink defines a particular *situation* in the address translation. Such situations can be referred from high-level descriptions of test programs, so-called *templates*. The latter are processed by the Genesys-Pro generator [2], which formulates constraints on instruction operands, solves them and transforms the results into the instruction sequences. The major advantage of the approach is the use of the highly developed languages for modeling address translation and describing test templates. The disadvantage is that the tool is not able to automatically extract *conflicts* and

dependencies between instructions. Verification engineers have to manually specify such kind of information in test templates.

In [5], the Java programming language coupled with a specialized library is used to specify MMU. As in DeepTrans, the situations correspond to the paths in the graph describing the subsystem under test; here is an example: $\{Mapped$ (data are requested via a mapped segment), $TLBHit$ (there is a TLB hit), $TLBValid$ (the matched TLB entry is valid), $\neg LIHit$ (a miss in the first-level cache occurs)}. In addition, the approach provides means for specifying instruction dependencies; an example is as follows: $\{\neg TLBEqual$ (instructions use different TLB entries), $LIIndexEqual$ (data are mapped to the same set of the first-level cache), $\neg LIEqual$ (data belong to different cache lines)}. Test templates are constructed automatically by combining situations and dependencies for short sequences of instructions. Building templates and creating programs on their basis is done by the MicroTESK generator (ISP RAS) [6]. The strength of the approach is systematic test enumeration that takes into consideration instruction execution paths as well as dependencies between instructions. The principal weakness is underdeveloped specification facilities.

IV. APPROACH DESCRIPTION

The main goal of the presented research is to combine the advantages of the methods [4] and [5] as well as to avoid their drawbacks. It can be achieved by using formal specifications. Accordingly, microprocessor instructions, an MMU and test templates are described in formal domain-specific languages. Specifications are analyzed to extract *testing knowledge*, that is, situations and dependencies. The information having been extracted is used to automatically generate test programs from templates as well as to automatically construct templates in a systematic way. The suggested method is supported by the MicroTESK TPG [7].

A. Formal Specifications

Formal specification of a microprocessor under test touches on the instruction set and the memory subsystem. Instructions are described in the nML language [8]. Descriptions declare the registers and define the assembly syntax, binary image and the semantics of the instructions. Semantics is specified in the usual imperative form by means of the bit-vector and floating point operations. Here is an nML specification of the MIPS [9] integer addition instruction (*ADD*):

```

op ADD (rd: REG, rs: REG, rt: REG)
  syntax = format("add %s, %s, %s",
    rd.syntax, rs.syntax, rt.syntax)
  image = format("000000%s%s%s00000100000",
    rs.image, rt.image, rd.image)
  action = {
    temp = rs<31>::rs<31..0> +
      rt<31>::rt<31..0>;
    if temp<32> != temp<31> then
      exception("IntegerOverflow");
    else
      rd = coerce(DWORD, temp<31..0>);
    endif;
  }

```

Being rather simple, nML does not have adequate facilities to describe memory management. Though the language is powerful enough to specify caching and address translation mechanisms, pure nML specifications of MMU are awkward and hardly analyzable; in particular, it is difficult to extract testing knowledge to automate test program generation. In that situation, a domain-specific language has been introduced. A memory access instruction is described in nML in an intuitive manner by reading or writing data from or to the byte array representing the physical memory. Every access to the array triggers the MMU logic specified in a separate file. An nML specification of the MIPS load byte instruction (*LB*) may look as follows:

```
op LB (rt: REG, offset: SHORT, base: REG)
  syntax = format("lb %s, %d(%s)",
    rt.syntax, offset, base.syntax)
  image = format("100000%s%s%s",
    base.image, rt.image, offset)
  action = {
    rt = MEM[base + offset];
  }
```

where *MEM* is an array declared as *mem MEM[2**36, BYTE]*; 2^{36} (that is 2^{36}) is the memory size in bytes. Note that notwithstanding the array is specified as the physical memory, it is accessed through the virtual address.

Memory management is described in a special language. MMU specifications include address types, memory segments, buffers, such as TLB and caches, and detailed algorithms for handling load and store instructions. Addresses and segments are described straightforwardly; buffers are specified with the following parameters: the *associativity (ways)*, the *number of sets (sets)*, the *entry (line) format (entry)*, the *index calculation function (index)*, the *tag calculation function (tag)* and the *data eviction policy (policy)*. Here is a description of the virtual and physical addresses (*VA* and *PA* correspondingly), user segment (*XUSEG*), address translation buffer (*TLB*) and the first-level cache memory (*L1*) of a MIPS microprocessor:

```
address VA (64)
address PA (36)

segment XUSEG (va: VA)
  range = (0x0, 0x00fffffffffff)

buffer TLB (va: VA)
  ways = 64
  sets = 1
  entry = (VPN2: 27, V0: 1, PFN0: 24, ...)
  index = 0
  tag = va<39..13>
  policy = NONE

buffer L1 (pa: PA)
  ways = 4
  sets = 128
  entry = (TAG: 24, DATA: 256)
  index = pa<11..5>
  tag = pa<35..12>
  policy = LRU
```

Processing of loads and stores is specified by requesting the buffers and handling their responses. The syntax is similar to nML though allows using such conditions as *XUSEG(va).hit* (the address *va* belongs to the segment *XUSEG*) and *L1(pa).hit* (the buffer *L1* contains the data for the address *pa*). Here comes an example:

```
mmu MEM (va: VA)
...
read = {
  if XUSEG(va).hit then
    if TLB(va).hit then
      tlbEntry = TLB(va);
    else
      exception("TLBRefill");
    endif;
    if va<12> == 0 then
      v = tlbEntry.V0;
      pfn = tlbEntry.PFN0;
      ...
    endif;
    if v == 1 then
      pa = pfn::va<11..0>;
    else
      exception("TLBInvalid");
    endif;
    ...
  endif;
  if L1(pa).hit then
    l1Entry = L1(pa);
    data = l1Entry.DATA;
    ...
  endif;
}

write = { ... }
```

B. Coverage Extractor

Formal specifications are parsed and the *control flow graph (CFG)* is build. A *coverage extractor* traverses the CFG and constructs the set of all possible execution paths (the graph is assumed to be acyclic). A single path, so-called a *situation*, describes processing of an individual request and finishes either with a memory access or with an exception (incorrect address, TLB refill, etc.). Each transition of the path is labeled with a *guard*, i.e. a condition that enables the transition, and an *action* to be performed. Here is an example of a load situation (for the sake of simplicity, the transition actions are omitted): *{XUSEG(va).hit, TLB(va).hit, va<12> = 0, v = 1, L1(pa).hit}*.

Given a pair of execution paths, the coverage extractor may be demanded to construct the set of all possible *dependencies*. A dependency is a map from the set of buffers common for the two given execution paths to the set of *conflicts*. Speaking formally, a dependency is a partial map $d: B \rightarrow C$, where B is the set of buffers and C is the set of conflicts. The following types of buffer usage conflicts are predefined in the tool:

- *AddrEqual* – using the same data;
- *AddrNotEqual* – using different data:
 - *IndexEqual* – using data of the same set:
 - *TagEqual* – using data of the same line;
 - *TagReplaced* – using data of the replaced line;
 - *TagNotReplaced* – otherwise;
 - *IndexNotEqual* – using data of different sets.

To illustrate the concept, let us consider two simple situations: the first one is *{..., TLB(va₁).hit, ..., L1(pa₁).hit}*; the second is *{..., TLB(va₂).hit, ..., L1(pa₂).miss, ...}*. The situations share two buffers, namely TLB and L1. A possible dependency is *{TLB.TagEqual, L1.IndexNotEqual}*, that is, two instructions access the same TLB entry ($va_1<39..13> = va_2<39..13>$), but use different L1 sets ($pa_1<11..5> \neq pa_2<11..5>$).

C. Template Iterator

A *template* is a sequence of situations linked together with a number of dependencies. A *template iterator* systematically enumerates templates to cover a representative set of cases of the memory subsystem behavior. Let S be the set of situations; D be the set of dependencies; n be the length of templates. Formally, a test template of the length n is a pair $\langle \sigma, \lambda \rangle$, where $\sigma = (s_1, \dots, s_n) \in S^n$ is the *template skeleton* and $\lambda = \{d_{ij}\}$, where $i = 1, \dots, n-1$ and $j = i+1, \dots, n$, is the *template ligaments*. An example of a two-situation template is given below:

$s_1: \{XUSEG(va_1).hit, TLB(va_1).hit, va_1<12> = 1, v_1 = 1, LI(pa_1).hit\};$
 $s_2: \{XUSEG(va_2).hit, TLB(va_2).hit, va_2<12> = 0, v_2 = 0\};$
 $d_{12}: \{TLB.TagEqual (va_1<39..13> = va_2<39..13>)\}.$

The main, but not the only, approach supported by the tool is combinatorial generation. Test templates are constructed by enumerating all possible skeletons of the given length and creating all possible ligaments for each of them. The template iterator checks whether the produced templates are consistent. For each template, it formulates the set of constraints and invokes a solver [10]; if the constraints are unsatisfiable, the template is discarded. Here is an example of an inconsistency:

$s_1: \{..., va_1<12> = 0, v_1 = 1, ...\};$
 $s_2: \{..., va_2<12> = 0, v_2 = 0\};$
 $d_{12}: \{TLB.TagEqual (va_1<39..13> = va_2<39..13>)\}.$

$TLB.TagEqual$ implies that both instructions access the same TLB entry, whereas $va_1<12> = 0$ and $va_2<12> = 0$ result in $v_1 = v_2 = tlbEntry.V0$, which contradicts to $v_1 = 1$ and $v_2 = 0$.

To avoid the combinatorial explosion, special heuristics are in use. Among them, *factorization of situations* and *limitation of the depth of dependencies* are essential. Description of the heuristics are out of the scope of the paper.

D. Test Data Generator

Templates are symbolic representation of test programs. To produce a test program from a template, the latter should be instantiated. A *test data generator* plays the key role in this activity. Test data, in a sense, are a solution to the constraints stipulated in the template. They include virtual addresses to be used by the instructions as well as some auxiliary information intended for setting up the state of the microprocessor under test such as indices of TLB entries, VPN-to-PFN mappings, sequences of addresses to be accessed to load or evict data to or from the buffers, etc.

The test data generator acts in compliance with one of the following strategies: (1) *heavyweight* template elaboration with an attempt to find an exact solution to the problem or (2) *lightweight* processing targeted at constructing an approximate solution. In the main, our approach follows the second strategy. Detailed analysis of templates makes sense only for accurate MMU specifications, while instruction-level models are rather abstract. Another argument is that the lightweight approach gives a significant benefit in terms of performance, while the quality of testing is comparable.

Given a template $\langle (s_1, \dots, s_n), \{d_{ij}\} \rangle$, consider how test data are generated. First, for each situation s_j of the template, a *united dependency* $\mathbf{dep}_j: B \times C \rightarrow 2^{\{1, \dots, j-1\}}$ is built. For each

buffer b and conflict c , $\mathbf{dep}_j(b, c)$ contains indices $i < j$ such that $b \in \mathbf{dom}(d_{ij})$ and $d_{ij}(b) = c$, that is, the situations s_i and s_j access the buffer b and there is the access conflict c . Then, the template's situations are processed one after another. Given a situation s_j , the buffers affected in s_j are sequentially inspected. For each buffer b , the actions listed below are performed:

- if $\mathbf{dep}_j(b, AddrEqual) \neq \emptyset$, then
 $\mathbf{data}(s_j).addr \leftarrow \mathbf{data}(s_i).addr$,
 where $\mathbf{data}(s_j)$ denotes the test data associated with s_j ; $addr$ is the virtual or physical address depending on the b type; i is any index from $\mathbf{dep}_j(b, AddrEqual)$;
- otherwise, if $\mathbf{dep}_j(b, IndexEqual) \neq \emptyset$, then
 $\mathbf{data}(s_j).addr<I> \leftarrow \mathbf{data}(s_i).addr<I>$,
 where I is the bit range given in the index section of the b specification;
 - if $\mathbf{dep}_j(b, TagEqual) \neq \emptyset$, then
 $\mathbf{data}(s_j).addr<T> \leftarrow \mathbf{data}(s_i).addr<T>$,
 where T is the bit range given in the tag section of the b specification;
 - if $\mathbf{dep}_j(b, TagReplaced) = \emptyset$, then
 $\mathbf{data}(s_j).addr<T> \leftarrow \mathbf{tag}_b(\mathbf{data}(s_i).addr<I>)$,
 where $\mathbf{tag}_b(index)$ is a previously unused tag of b for the given index;
- otherwise (if $\mathbf{dep}_j(b, IndexEqual) = \emptyset$),
 $\mathbf{data}(s_j).addr<I> \leftarrow \mathbf{index}_b$,
 where \mathbf{index}_b is a previously unused index of b .

TagReplaced conflicts – referred to as *dynamic conflicts* – are handled in a special way. As soon as all other constraints, including hits and misses (see the next paragraph for details), are resolved, the created sequence of instructions is simulated on a simplified model derived from the MMU specifications. This enables the generator to predict the lines being evicted and replaced with recently accessed data. If there is a *TagReplaced* conflict between two instructions (template situations, to be more precise), the evicted tag having been predicted for the first instruction is copied into the address of the second one.

In between static *Equal/NotEqual* and dynamic *Replaced* conflicts, hits and misses are considered. For a hit, an access to the designated address is appended to the template test data: $\mathbf{hit}(b).add(\mathbf{data}(s_j).addr)$, where $\mathbf{hit}(b)$ is a set-separated data structure that stores sequences of addresses targeted at loading data into the buffer b . For a miss, an address sequence ω is added: $\mathbf{miss}(b).add(\omega)$, where $\mathbf{miss}(b)$ is a storage of addresses used to evict data from b , and $\omega = \{addr_1, \dots, addr_W\}$ is a so-called *evicting sequence*, that is, $addr_k<I> = \mathbf{data}(s_j).addr<I>$, $addr_k<T> \neq \mathbf{data}(s_j).addr<T>$ and $addr_k<T> \neq addr_l<T>$ for all $k, l \in \{1, \dots, W\}$ such that $k \neq l$; W is the b associativity. Note that appending an address to the $\mathbf{hit}(b)$ structure may require adding evicting sequences for the preceding buffers with the miss constraint having been set.

E. Test Data Adapter

Indeed, test data concretize symbolic templates, but being instruction set independent they are still too general to be immediately applied to testing. It is a *test data adapter* who

translates a template coupled with test data into a sequence of specific instructions, so-called a *test case*. Such a sequence usually consists of two parts: a *preparation*, which sets up the microprocessor state, and a *stimulus*, which performs a series of memory accesses to stress the microprocessor's MMU.

Making a stimulus is straightforward: each situation of the template skeleton is converted into a load or a store depending on the specification section, *read* or *write*, the execution path belongs to. A particular type of the instruction, i.e. the size of a data block being accessed, is either derived from the template / specifications or randomized. The instruction is allowed to use any registers from the user-defined set. Note that the procedure requires a mapping from $\{read, write\} \times \{byte, word, \dots\}$ to the set of memory access instructions implemented in the design.

Constructing a preparation sequence is more intricate. The main problem is that placing data into a buffer may change the state of others. Here is how the problem is solved. First, virtual address based buffers, e.g., TLB, are handled before buffers accessed by physical addresses, e.g., L1 and L2. Initialization of the latter can be carried out by using unmapped addresses, which does not affect the former. Second, the "largest buffer first" strategy is applied. Typically, a set of lines of a smaller buffer maps several sets of lines of a larger one, which gives a possibility to change the smaller buffer with no tangible effect to the larger one. Given a buffer, the preparation sequence is cut into pieces corresponding to particular sets of the buffer. Each piece is the catenation of the **miss** and **hit** sequences. It is implied that each buffer is provided with a code pattern to be used to place data for a given address. Here comes a simplistic test case for the MIPS architecture:

```
// Preparation:
// Fill TLB: VPN0=0x4, V0=1, PFN0=0x10222
tlbwi ...
// Fill L1: VA=0x80261026 (PA=0x261026)
lui t0, 0x8026
ori t0, t0, 0x1026
lb t0, 0(t0)
// Address 0: VA=0x80261026 (PA=0x261026)
lui s0, 0x8026
ori s0, s0, 0x1026
// Address 1: VA=0x4059 (PA=0x10222059)
ori s1, zero, 0x4059

// Stimulus:
// KSEG0.hit (Mapped=0), L1.hit
lb a0, 0(s0)
// XUSEG.hit (Mapped=1), TLB.hit, VA[12]=0, V=1
sb a1, 0(s1)
```

The instructions here are as follows [9]: *TLBWI* writes a TLB entry; *LUI* loads a constant into an upper half of a word; *ORI* does a bitwise OR with a constant; *LB* loads a byte from memory; *SB* stores a byte to memory.

Preparations may be of significant length, but the tool is able to reduce the volume of such kind of code. It keeps track of the microprocessor state during test generation and skips useless initialization (e.g., it does not load data into a buffer if they are already there). Moreover, the generator can choose a data tag so as to fit the desired event, a hit or a miss. On the other hand, preparation sequences are of interest as they – as our experience shows – can stress the memory subsystem and discover "high-quality" bugs.

V. INDUSTRIAL APPLICATION

The proposed approach is implemented in the MicroTESK test program generator [6, 7]. Since 2006, different versions of the tool – including one described in [5] – have been applying to functional verification of several industrial microprocessors with the MIPS architecture [9]. MMU specifications take into account such buffers as a JTLB (a joint TLB), a DTLB (a micro TLB used to speed up data address translation), an L1 (a first-level cache) and an L2 (a second-level cache). Besides, they involve mapped and unmapped memory segments (XUSEG, KSEG0, KSEG1 and XKPHYS), TLB control bits (Valid, Dirty and Global) and cache policies (various combinations of Write-Through, Write-Allocate and Write-Back flags). Stimuli are composed from load and store instructions. The approach has allowed revealing a great number of critical bugs (e.g., reading incorrect data from memory) in the MMU designs, which had not been detected by randomly generated test programs.

VI. CONCLUSION

Functional verification of a microprocessor MMU is surely a hard nut to crack. Automation facilities are undoubtedly of high value and importance. Our work contributes its mite to improving verification quality and productivity. The proposed solution is based on the memory subsystem specification, i.e. on formal descriptions of caching and address translation. The distinctive features of the approach are high automation and systematicness. The suggested method is implemented in the MicroTESK test program generator, which is freely distributed open-source software. The tool has been used and is being used in industrial projects on microprocessor development. A bad news is that the recent release has no support for multicore designs. Avoiding this shortcoming is a priority task for the nearest future. More particularly, we are going to extend the approach to multiprocessor systems with distributed memory.

REFERENCES

- [1] Bryant R.E., O'Hallaron D.R. Computer Systems: A Programmer's Perspective. Pearson, 2010. 1080 p.
- [2] Adir A., Almog E., Fournier L., Marcus E., Rimov M., Vinov M., Ziv A. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. *Design & Test of Computers*, 2004. pp. 84-93.
- [3] Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. *Morgan and Claypool*, 2011. 195 p.
- [4] Adir A., Fournier L., Katz Y., Koifman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. *High-Level Design Validation and Test Workshop*, 2006. pp. 102-110.
- [5] Vorobyev D., Kamkin A. Generatsiya testovykh programm dlya podsystemy upravleniya pamyat'yu mikroprotssora [Test Program Generation for Memory Management Units of Microprocessors]. *Trudy ISP RAN [The Proceedings of ISP RAS]*, 2009, vol. 17. pp. 119-132 (in Russian).
- [6] Kamkin A., Tatarnikov A. MicroTESK: An ADL-Based Reconfigurable Test Program Generator for Microprocessors. *Spring/Summer Young Researchers' Colloquium on Software Engineering*, 2012, pp. 64-69.
- [7] MicroTESK page — <http://forge.ispras.ru/projects/microtesk>
- [8] Freericks M. The nML Machine Description Formalism. Technical Report TR SM-IMP/DIST/08, TU Berlin CS Department, 1993.
- [9] MIPS64™ Architecture For Programmers. *MIPS Technologies Inc.*
- [10] Fortress page — <http://forge.ispras.ru/projects/solver-api>

An approach to Direct Memory Access module verification

Aleksey Meshkov
MCST

Russia, Moscow, Vavilova 24
Email: alex@mcst.ru

Mikhail Ryzhov
MCST

Russia, Moscow, Vavilova 24
Email: ryzhov@mcst.ru

Pavel Frolov
MCST

Russia, Moscow, Vavilova 24
Email: opium@mcst.ru

Abstract—A method of direct memory access subsystem verification used for Elbrus series microprocessors has been described. A peripheral controller imitator has been developed in order to reduce verification overhead. The model of imitator has been included into the functional machine simulator. A pseudorandom test generator for verification of the direct memory access subsystem has been based on the simulator.

I. INTRODUCTION

Modern computer systems require very intensive data exchange between the peripheral devices and the random-access memory. In the most cases this exchange is performed by the direct memory access (DMA) subsystem. The increasing demands for the performance of the subsystem lead to an increase in its complexity, therefore requiring development of effective approaches to DMA subsystem verification [1], [2].

This article is based on a result of a comprehensive project than combined implementation of a there co-designed verification techniques based on the consecutive investigation of the DMA subsystem employing one the three models: 1) a functional model written in C++ that corresponds to behaviour of the subsystem in the environment determined by a real computer system configuration, 2) RTL model in Verilog and 3) FPGA-based prototype. This article describes the first method that enables verifying correctness of the design at an early stage of the verification and eliminate a large quantity of bugs using simple tests.

The most important problem that significantly affects the quality of the subsystem verification is the exhaustiveness of the representation of the external devices connected to it and input vectors they generate. In this case, the problem has been solved by introducing a device imitating a peripheral controller and capable of generating a comprehensive range of DMA subsystem interaction patterns into the functional model. The basic aspects of DMA imitator implementation are presented in the second section.

The exhaustiveness of the subsystem in question verification is achieved with a test generator allowing to provide necessary inputs using the imitator. The generator produces a test program that performs the DMA imitator scenarios setup for all of its agents, launches their concurrent execution, provides memory access by the CPU cores during the DMA access scenarios execution and checks the final memory state. The generator operation principles are described in the fourth section of the paper.

The generation of final memory state checking code requires a golden model of the memory subsystem being available for the generator. A functional model library that will be described in the third section has been reused from previous projects in order to fulfill this requirement.

II. PERIPHERAL DEVICE IMITATOR

Considering the computer system containing the subsystem (fig. 1a) in question it should be noted that difficulties connected to precise modeling of the southbridge devices caused by the usage of the complex device drivers can be avoided via imitating behavior of the real DMA agents. A masked DMA copy operation has been used as a basic operation that allows to implement the significant number of the direct memory access scenarios. In order to achieve a high-speed test execution, the imitator is integrated into to IO link between the northbridge and the chipset (southbridge, fig. 1b). The positioning of the imitator as a standard IO controller allows to apply this scheme to any modern Elbrus series processor.

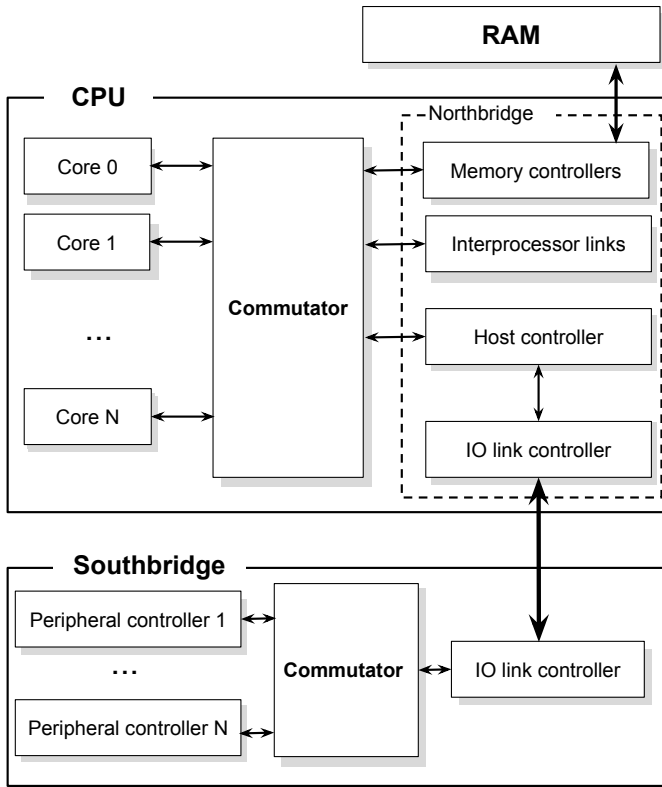
The imitator represents a simplified version of the southbridge. It includes adjustable number of identical agents (fig. 3), each capable of working in normal or table modes. In the table mode the memory access scenario specification is simplified by providing them via tables placed in the memory.

Agent is capable of the following operations:

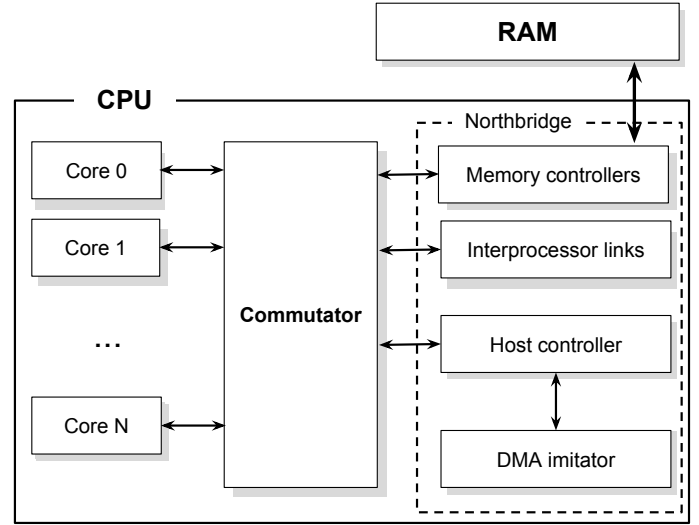
- copying data from one area of the memory to another in normal and table modes,
- reading copy operation parameters from memory,
- data transformation.

The imitator is implemented as a PCI-compatible device, each agent is created as an independent device that is controlled by a common bus via load and store operations to the configuration space. Agents can perform an exchange with the memory using standard read and write packets. The commutation between the agents is performed by the DMA Switch module.

The structure of the DMA-agent is shown at this fig. 3. `ConfigResigters` module is an array of configuration space registers containing setup operation modes, base addresses and other parameters. In the normal mode the addresses are written to the `ConfigRegisters` are used to access the memory. In the table mode the `TMHandler` module uses written address to fetch and process the table with address



(a) Real configuration



(b) Model configuration (integration of the DMA imitator into the northbridge)

Fig. 1. The structure of the computer systems:

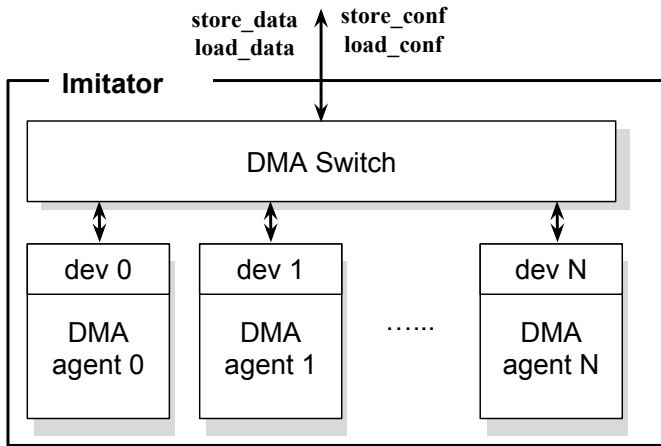


Fig. 2. The structure of the DMA-imitator

of reads and writes. The **Format** module is responsible for masking the data and correct merging of data in the table mode. The **DMAEngine** module implemented as a FIFO buffer with data performs loads and stores of the data using the DMA write and DMA read functions provided by the functional model.

III. FUNCTIONAL MODEL OF THE DMA IMITATOR

The approach to the problem is based on presenting the direct memory access as two independent modules: the simu-

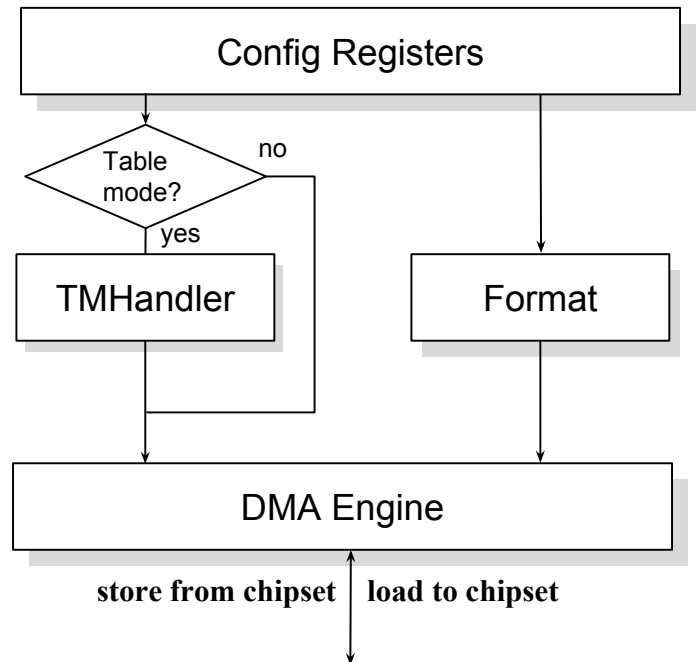


Fig. 3. The DMA-agent

lator, that imitates the work the computer system architecture objects that are directly employed in the process, and a test generator that provides the modes and parameters for the direct

memory access, sets up the logic of these objects and controls the correctness of the outcome (fig. 4). The structural and functional independence of these modules significantly increases the flexibility of the system in such aspects as content and interaction of objects under study, the spectrum of generated inputs and results checking.

The configuration of the simulator that has been developed contains four processors each one containing several general-purpose cores and a northbridge, the southbridge and an imitator that consists of an array of peripheral devices and their interfaces [3]. According to the second section the communications of the imitator and the northbridge are performed by the functions of the programming model described in the PCI standard.

The simulator works according to interpretation principle. In each virtual tick execution of one command in each of the processor cores is performed. In addition, different asynchronous actions in respect to the commands execution actions such as counter and timer ticks and external interrupt handling are also performed during a single tick.

In order to enable the communication of the simulator with the generator it has been decided to implement a working cycle of the simulator available through a set of library functions.

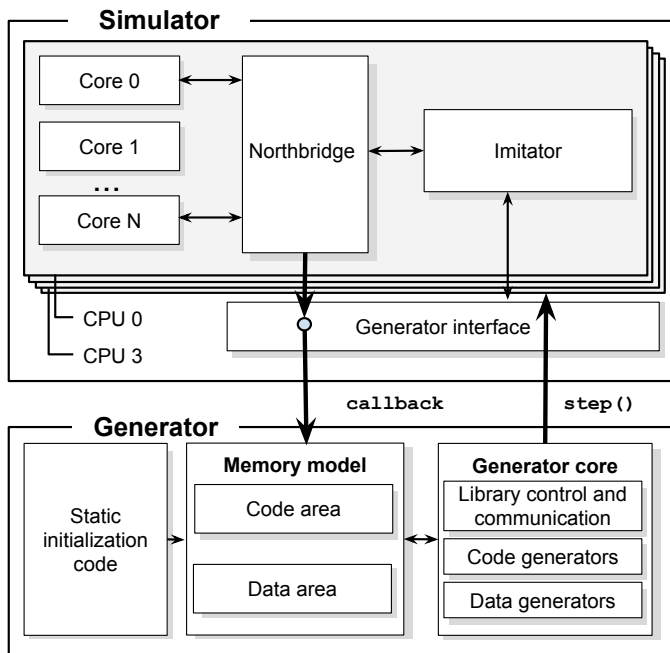


Fig. 4. Components of the DMA subsystem functional model

IV. TEST GENERATOR

The generator contains the static initialization code, the memory model and the core of the generator. The initialization code is a sequence of instructions that performs the initial setup of the hardware performed by the test.

The core of the generator contains the library control and communication module as well as the code and data generators [5]. The library control and communication module is responsible for interaction with the simulator. It invokes the

`step()` function that implements execution of instructions of the modeled hardware and the analysis the result of its execution. The code generator writes the code that controls the operation of each of the DMA-agents and the data generator writes the blocks of the data to be send. The flexibility of the DMA-imitator parameterization is fully supported by the pseudorandom test generator that sets up pseudorandom parameters for the DMA-exchange such as addresses of the memory buffers, ranges of the DMA-packet sizes as well as different transfer modes.

Both static initialization code and dynamically generated code is placed into the code area that is one of the components of the memory model. When code fetch takes place during the program execution the requests are directed by the callback function to the code area of the generator. The data area that is another memory model component is handled in a similar manner. The requests for the data — the loads and stores can be initiated by both the CPU cores and the DMA-agents. All of the requests are redirected to the data structure containing the array dynamically allocated by the data generator.

The step-by-step algorithm of the simulator main modules interaction with the generator is presented in the fig. 5.

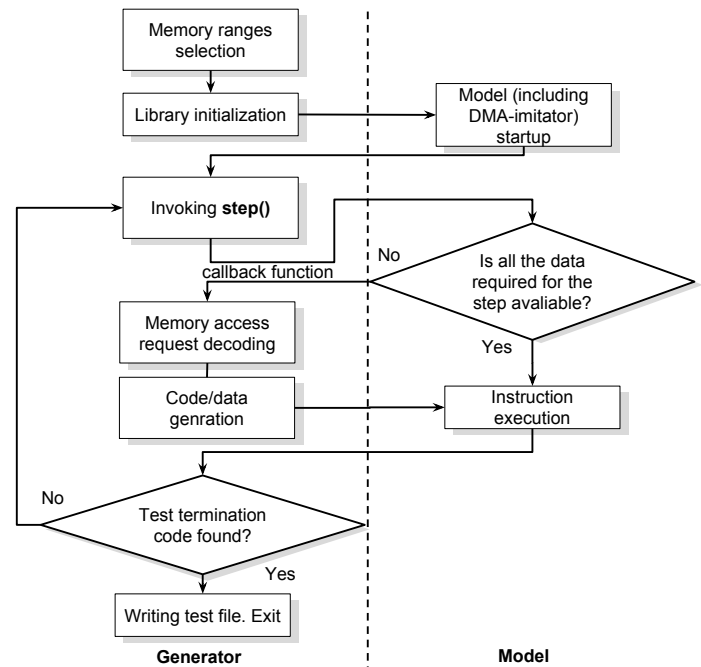


Fig. 5. The control flow of the generator that employs DMA subsystem functional model

The general scenario of working with the DMA-imitators has the following outline: the basic system initialization, the initialization of the DMA buffers with the data designated for transmission, the configuration of the DMA-imitator and the launch of the DMA-exchange. Different system parameters such as number of processors and available physical address ranges can be varied in a random way to create different DMA routing scenarios. The system initialization procedure can also turn on input/output memory management unit (IOMMU) and fill its translation table with random entries.

The initialization of the DMA buffers is performed by the CPU cores causing the data for the transfer to be located at different levels of the coherent memory hierarchy that includes both caches and memory [6]. During the configuration of the imitator the specification of the operation mode and the base address of the memory to be processed are determined. The DMA exchange is performed while the CPU cores access memory regions that intersect with the DMA buffers. After the completion of the exchange the reference values are generated based on the contents of the memory final state. These values are used to perform self-checking during test execution on the target model or device.

Any test produced by the generator can be executed on either the RTL model, the simulator or the FPGA-based prototype without any additional test modification. The test generator provides an opportunity to use any device connected to real southbridge instead of the DMA imitator such an ethernet controller as a source of DMA-packets.

V. FUTURE WORK

In this work the basic infrastructure for the DMA subsystem verification was developed and implemented. The further work is supposed to be focused on the test generator elaboration. There is a need to customize the test generation parameters to reproduce specific test situations (corner cases), actual for the verification not only of the DMA subsystem, but also of system caches, IOMMU, interrupts and so on in the presence of the DMA activity. The generator must support arbitrary test generation scenarios, which can define certain parameter constraints for the random generation as well as test code patterns. The test generation scenarios support is under development.

VI. CONCLUSION

In this study the problem of the direct memory subsystem verification when applied to “Elbrus” series microprocessors has been investigated. In order to enable the execution of sufficient number of tests and speeding up the development of the test generators and bug analysis a method of verification based on the replacement of DMA-capable real devices with imitator device with a simple programming interface and ability to completely consume the bandwidth of the direct memory access data path was introduced. The application of the developed method enables to achieve the operation modes of the DMA subsystem analogous to the real-world ones. The unification of the DMA imitator interface for the RTL-model, the computer complex simulator and the FPGA-based prototype allows to increase the pace of DMA subsystem tests generator development.

REFERENCES

- [1] Grosso, M. et al. Functional Verification of DMA Controllers - Journal of Electronic Testing: Theory and Applications Volume 27 Issue 4, August 2011, Pages 505-516.
- [2] A.K. Kim, M.S.Mikhailov, V.M.Fel'dman. Podsystema vvoda-vyvoda dlya sistem na kristalle “MCST-4R” i “Elbrus-S” na osnove mikroskhemy kontrollera periferiinykh interfeisov. Voprosy radioelektroniki, seriya EVT, vypusk 3, 2012.
- [3] Gurin K.L., Meshkov A.N., Sergin A.V., Yakusheva M.A. Razvitie modeli podsystemy pamyati vychislitel'nykh kompleksov serii El'brus. Voprosy radioelektroniki, seriya EVT, 2010, vypusk 3.
- [4] Nohl, A., Braun, G., Schkieber, O., Leupers, R., Meyr, H., A Universal Technique for Fast and Flexible Instruction-Set Architecture Simulation, DAC2002, June 10-14, New Orleans, Louisiana, USA, 2002.
- [5] Frolov P.V. Generatsiya sluchainykh testov sistemnogo urovnya dlya mikroprotessorov s arkhitekturoi El'brus. Voprosy radioelektroniki, seriya EVT, 2014, vypusk 3.
- [6] Isaev M.V., Polyakov N.Yu. Primenenie kesha i spravochnika DMA-obmenov v NUMA-sistemakh dlya povysheniya proizvoditel'nosti podsystemy vvoda-vyvoda. Pervaya vserossiiskaya nauchno-tehnicheskaya konferentsiya Raspletinskie chteniya : sb. tez. dokl. Moskva, 2013. S. 169-170.

An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs

Igor Melnichenko
INEUM
Moscow, Russian Federation
igor.melnitxenko@gmail.com

Alexander Kamkin, Sergey Smolov
ISP RAS
Moscow, Russian Federation
{kamkin, smolov}@ispras.ru

Abstract—Model-based test generation is widely spread in functional verification of hardware designs. The extended finite state machine (EFSM) is known to be a powerful formalism for modelling digital hardware. As opposed to conventional finite state machines, EFSM models separate datapath and control, which makes it possible to represent systems in a more compact way and, in a sense, reduces the risk of state explosion during verification. In this paper, a new EFSM-based test generation approach is proposed and compared with the existing solutions. It combines random walk on a state graph and directed search of feasible paths. The first phase allows covering “easy-to-fire” transitions. The second one is aimed at “hard-to-fire” cases; the algorithm tries to build a path that enables a given transition; it is carried out by analyzing control and data dependencies and applying symbolic execution techniques. Experiments show that the suggested approach provides better transition coverage with shorter test sequences comparing to the known methods and achieves a high level of code coverage.

Keywords—hardware design; hardware description language; simulation-based verification; test generation; modelling; extended finite state machine; graph traversal; random walk; backjumping; symbolic execution; constraint solving

I. INTRODUCTION

Functional verification is a labor-intensive and time-consuming stage of the hardware design process. According to [1], it spends about 70% of the effort, while the number of verification engineers is usually twice the number of designers. Moreover, the “verification gap”, i.e. a difference between verification needs and capabilities, seems to grow over time [2]. In such a situation, improvement of the existing verification methods and development of new ones is of high value and importance. *Simulation-based verification*, often referred to as *testing*, is a widely accepted approach to hardware verification. It requires a *testbench* [1], a special environment that generates *inputs*, so-called *stimuli*, *vectors* or *patterns*, and optionally observes the *outputs*, so-called *reactions*.

Among the methods for stimulus generation, *model-based approaches* are of interest. Being formal representations of designs under test, models serve as a valuable source of “testing knowledge”. There are a lot of model types used for specifying hardware: *finite state machines (FSM)* [3], *extended*

FSM (EFSM) [4], *Petri nets* [5], etc. The key distinction of the EFSM formalism is clear separation of data and control flows. It is worth mentioning that EFSM models can be automatically extracted from HDL descriptions making it possible to generate code coverage-directed tests [6].

This article advances the FATE approach to EFSM-based functional test generation (FTG) [7]. The main feature of FATE is *backjumping*: if an EFSM traverser fails to cover a transition, it tries to detect a cause of the failure (that is, a transition which must be traversed in order to enable the target one) and constructs a path directly from the found transition. Another important part of the approach is a special heuristic addressing *counters* and *loops*. However, FATE is hardly applicable to hardware designs with complicated data and control dependencies.

The rest of the paper is organized as follows. Section II defines the EFSM model and briefly describes an EFSM extraction method having been used. Section III considers the original FATE approach, while Section IV introduces a number of improvements to it. Section V proposes a new EFSM-based FTG method and shows how it works by the example of two simple EFSMs. Section VI contains an experimental comparison of the abovementioned approaches. Section VII concludes the paper and outlines directions for future improvement of the suggested algorithm.

II. EFSM MODEL AND HDL-TO-EFSM EXTRACTION

Let V be a set of *variables*. A *valuation* is a function that associates each variable with a value from the corresponding domain. The set of all valuations over V is denoted as D_V . A *guard* is a Boolean function defined on valuations ($D_V \rightarrow \{true, false\}$). An *action* is a transformation of valuations ($D_V \rightarrow D_V$). A pair $\gamma \rightarrow \delta$, where γ is a guard and δ is an action, is called a *guarded action*. When we speak about a function, it is implied that there is a *description* of the function in some formal language (thus, we can reason about the function’s syntax, not only the semantics).

An EFSM is a tuple $M = \langle S_M, V_M, T_M \rangle$, where S_M is a set of *states*, $V_M = (I_M \cup O_M \cup R_M)$ is a set of *variables*, consisting of *inputs* (I_M), *outputs* (O_M) and *registers* (R_M), and T_M is a set of *transitions* (all sets are supposed to be finite). Each transition

$t \in T_M$ is a tuple $(s_t, \gamma_t \rightarrow \delta_t, s'_t)$, where s_t and s'_t are respectively the *initial* and the *final state* of t , whereas γ_t and δ_t are respectively the *guard* and the *action* of t . A valuation $v \in D_{VM}$ is referred to as a *context*, while a pair $(s, v) \in S_M \times D_{VM}$ is called a *configuration*. A transition t is said to be *enabled* for a configuration (s, v) if $s_t = s$ and $\gamma_t(v) = \text{true}$.

Given a *clock* C (a periodic event generator) and an *initial configuration* (s_0, v_0) , the EFSM operates as follows. In the beginning, it resets the configuration: $(s, v) \leftarrow (s_0, v_0)$. On every tick of C , it computes the set of enabled transitions: $E \leftarrow \{t \in T_M \mid s_t = s \wedge \gamma_t(v) = \text{true}\}$. A single transition $t \in E$ (chosen nondeterministically) fires; the EFSM changes the configuration (updates the context and moves from the initial state to the final one): $(s, v) \leftarrow (s'_t, \delta_t(v))$.

In this paper, we do not discuss in detail the way the EFSM models are extracted. At the experimental phase, we use an implementation of the method introduced in [8]. The method deals with HDL descriptions written in synthesizable subsets of VHDL and Verilog [9]. The major advantage of the approach is high automation – it requires no information except HDL code. The method uses heuristics for identifying states and clock signals and extracts the EFSM from the control flow graph-based representation. For every process defined in the HDL description, a single EFSM is usually built; all EFSM models of the description are defined over the same set of variables. It should be emphasized that EFSM actions have the “flat” syntax, which means that each action is a linear sequence of assignments.

We have enhanced the cited method by adding a new heuristic aimed at recognizing the *initial configuration*. A guarded action $\gamma_r \rightarrow \delta_r$ is said to be *resetting* if the following properties hold: (1) γ_r depends on exactly one clock signal, which is called a *reset*; (2) δ_r consists solely of assignments of the kind $v = c$, where $v \in (O_M \cup R_M)$ and c is a constant expression. Provided that there is only one resetting action, that action is supposed to lead to the initial EFSM configuration.

III. THE ORIGINAL FATE ALGORITHM

The aim of the FATE algorithm is to generate a test that covers all transitions of a given multi-EFSM system. A *test* is a set of *test sequences*, i.e. sequences of test vectors. A *test vector* is a valuation over the joint set of the EFSMs' inputs. The algorithm includes three phases: an *EFSM analysis*, a *random traversal* and a *directed traversal*.

A. EFSM Analysis

In the beginning, for each EFSM of the system, data and control dependencies between its transitions are derived. Let t and τ be transitions and v be a variable. v is said to be *defined* in t ($v \in \text{Def}_t$) if δ_t contains an assignment to v ; v is said to be *used* in τ ($v \in \text{Use}_\tau$) if v appears either in γ_τ ($v \in \text{Use}_{\gamma_\tau}$) or in the right hand side of δ_τ ($v \in \text{Use}_{\delta_\tau}$). It is said that τ is *data dependent* on t (via v) if there exists a variable v such that $v \in (\text{Def}_t \cap \text{Use}_{\delta_\tau})$ and there exists a path $P = \{t_i\}_{i=1}^n$ from t to τ ($s'_t = s_{t_1}$ and $s'_{t_n} = s_\tau$) that does not define v . Note that if $v \in \text{Def}_\tau$, there should be δ_τ 's assignment with v in the right hand side that precedes the assignments to v . It is said that τ is

control dependent on t (via v) if there exists a variable v such that $v \in (\text{Def}_t \cap \text{Use}_{\gamma_\tau})$ and there exists a path from t to τ that does not define v .

The derived data and control dependencies are represented by the directed graphs whose vertices are the transitions and arcs are the dependencies. Thus, each EFSM is associated with two such graphs (one is for the control dependencies; another is for the data dependencies).

The second step of the analysis is *counter detection*. A register r is said to be a *counter* if there is a loop in the EFSM such that: (1) there is a transition t that defines r ; (2) r is defined recurrently (the current value depends on the previous one); (3) there is a transition t' that is control dependent on t via r . For each counter, all data dependency loops are saved.

B. Random Traversal

After the analysis, the random traversal phase is launched. The phase is parameterized with two values, L and N , where L is the length of a test sequence and N is the number of test sequences in the test. The random traversal is described by the following pseudo-code ($\{M_i = \langle S_i, V, T_i \rangle\}_{i=1}^m$ are the EFSMs being tested; *result* is the generated test):

```

result ← ∅
coverage ← ∅
while |result| < N ∧ coverage ≠ ∪i Ti do
  reset({Mi})
  sequence ← ∅
  while |sequence| < L do
    vector ← ∅
    for i ∈ {1, ..., m} do
      out ← {t ∈ Ti | st = si}
      while out ≠ ∅ do
        t ← choose(out)
        out ← out \ {t}
        constraint ← refine(γt, vector ∪ v)
        if isSAT(constraint) then
          vector ← vector ∪ solve(constraint)
          coverage ← coverage ∪ {t}
        break
      end
    end // while out
  end // for i
  apply(vector, {Mi})
  sequence ← sequence · {vector}
end // while sequence
result ← result ∪ {sequence}
end // while result

```

The pseudo-code above is based on the following functions: *reset*({M_i}) initializes the configurations of the models {M_i}; *choose*(T) returns a random item of the non-empty set T; *refine*(γ, v) replaces variables of the formula γ with their values according to the partial valuation v; *isSAT*(γ) checks whether the constraint γ is satisfiable; *solve*(γ) returns a valuation v such that γ(v) = 1; *apply*(v, {M_i}) assigns the inputs of the models {M_i} according to the partial valuation v and executes the enabled transitions (uninitialized inputs are randomized). The symbols s_i and v denotes respectively the current state of the model M_i and the context (shared among all models).

Being defined over the same set of variables, the EFSM models may affect each other while being co-executed. To minimize the influence, the following technique is applied. Each EFSM M_i is supplied with two parameters, F_i and A_i , where F_i is a constant inversely proportional to the number of inputs used in the M_i 's guards (the more such inputs M_i has, the more models are expected to be affected by M_i) and A_i is a so-called *aging factor* (initially set to zero). The sum $(F_i + A_i)$ is supposed to be the priority for choosing the model M_i . The priorities specify the order in which the models are handled (**for** $i \in \{1, \dots, m\}$ **do** ... **end**). The main idea with the aging factor is as follows. If test vector generation for M_i fails ($\text{isSAT}(\text{constraint})$ returns *false* for an outgoing transition), A_i is increased by a constant ΔA . Note that [7] has no particular definition of ΔA ; we use the value $\Delta A = \min_{i=1,m} F_i$. After the model selection loop, the aging factor of the most priority model is set to zero.

C. Directed Traversal

If there are uncovered transitions after the random traversal, FATE proceeds with the directed generation. Before describing the phase, let us make a remark. The procedure below, applies Dijkstra's algorithm for finding a shortest path in a graph [10]; it is assumed that an arc weight is the number of registers used in the transition's guard. The directed traversal is performed separately for each EFSM. Here is the pseudo-code (M is the EFSM being tested; *result* is the generated test):

```

targets  $\leftarrow T_M \setminus \text{coverage}$ 
while targets  $\neq \emptyset$  do
  t  $\leftarrow \text{choose}(\text{targets})$ 
  covered  $\leftarrow \text{false}$ 
  for prefix  $\in \text{reach}(M, s_i)$  do
    reset( $M$ )
    sequence  $\leftarrow \emptyset$ 
    for vector  $\in \text{prefix}$  do
      apply(vector,  $M$ )
      sequence  $\leftarrow \text{sequence} \cdot \{\text{vector}\}$ 
    end // for vector
    constraint  $\leftarrow \text{refine}(\gamma_i, v)$ 
    if  $\text{isSAT}(\text{constraint})$  then
      vector  $\leftarrow \text{solve}(\text{constraint})$ 
      apply(vector,  $M$ )
      sequence  $\leftarrow \text{sequence} \cdot \{\text{vector}\}$ 
      result  $\leftarrow \text{result} \cup \{\text{sequence}\}$ 
      coverage  $\leftarrow \text{coverage} \cup \{t\}$ 
      covered  $\leftarrow \text{true}$ 
      break
    end
  end // for prefix
  if  $\neg \text{covered}$  then
    if  $\neg \text{process}(M, t)$  then
      warning "The transition t cannot be reached"
    end
  end
  targets  $\leftarrow \text{targets} \setminus \{t\}$ 
end // while targets

```

Besides the auxiliary functions defined above, this pseudo-code uses $\text{reach}(M, s)$, which returns the set of known test sequences reaching the state s of the model M , and $\text{process}(M, t)$, which tries to cover the transition t of the model M by taking into account the control dependencies (it will be described later on).

Note that if *targets* includes transitions outgoing from the covered states, $\text{choose}(\text{targets})$ returns one of them; transitions whose initial states has not been reached are selected only if there are no others. Here is the description of $\text{process}(M, t)$:

```

registers  $\leftarrow R_M \cap \text{Use}_{\gamma_i}$ 
for reg  $\in \text{registers}$  do
  defines  $\leftarrow \{t \in T_M \mid \text{reg} \in \text{Def}_t\}$ 
  for def  $\in \text{defines}$  do
    for prefix  $\in \text{reach}(M, s_{\text{def}})$  do
      reset( $M$ )
      sequence  $\leftarrow \emptyset$ 
      for vector  $\in \text{prefix}$  do
        apply(vector,  $M$ )
        sequence  $\leftarrow \text{sequence} \cdot \{\text{vector}\}$ 
      end
      path  $\leftarrow \text{shortestPath}(M, s'_{\text{def}}, s_i)$ 
      path  $\leftarrow \text{path} \cdot \{t\}$ 
      if  $\text{isCounter}(\text{reg})$  then
        constraint  $\leftarrow \text{refine}(\gamma_{\text{def}}, v)$ 
        vector  $\leftarrow \text{solve}(\text{constraint})$ 
        apply(vector,  $M$ )
        sequence  $\leftarrow \text{sequence} \cdot \{\text{vector}\}$ 
        loop  $\leftarrow \text{processCounter}(M, s'_{\text{def}}, t, \text{reg})$ 
        if loop  $= \text{null}$  then
          return false
        end
        path  $\leftarrow \text{loop} \cdot \text{path}$ 
      else
        path  $\leftarrow \{\text{def}\} \cdot \text{path}$ 
      end
    covered  $\leftarrow \text{true}$ 
    for p  $\in \text{path}$  do
      if reg  $\notin \text{Def}_p \vee p = t$  then
        γ  $\leftarrow \gamma_p$ 
      else
        γ  $\leftarrow \gamma_p \wedge \gamma_{t|\text{reg}}[\delta_p]$ 
      end
      constraint  $\leftarrow \text{refine}(\gamma, v)$ 
      if  $\text{isSAT}(\text{constraint})$  then
        vector  $\leftarrow \text{solve}(\text{constraint})$ 
        apply(vector,  $M$ )
        sequence  $\leftarrow \text{sequence} \cdot \{\text{vector}\}$ 
      else
        covered  $\leftarrow \text{false}$ 
        break
      end
    end // for p
    if covered then
      result  $\leftarrow \text{result} \cup \{\text{sequence}\}$ 
      coverage  $\leftarrow \text{coverage} \cup \{t\}$ 
      return true
    end
  end // for prefix
end // for def
end // for reg
return false

```

The following notations are used: $\text{shortestPath}(M, s, s')$ finds the shortest path between the states s and s' of the M 's state graph using Dijkstra's algorithm; $\gamma_{|v}$ denotes the minimal sub-constraint of the constraint γ that depends on the variable v such that $\gamma \rightarrow \gamma_{|v}$ holds; $\gamma[\delta]$ stands for the constraint produced from γ by applying the substitution corresponding to the action δ . Here is the pseudo-code for $\text{processCounter}(M, s, t, \text{reg})$.

```

if  $\gamma_{t|reg}(v)$  then
  return {}
end
loop  $\leftarrow$  null
loopIterator  $\leftarrow$  init( $M, s, reg$ )
while  $\neg \gamma_{t|reg}(v)$  do
  while hasNext(loopIterator) do
    tempContext  $\leftarrow$  v
    tempSequence  $\leftarrow$  sequence
    loop  $\leftarrow$  next(loopIterator)
    for  $l \in$  loop do
      constraint  $\leftarrow$  refine( $\gamma_l, v$ )
      if isSAT(constraint) then
        vector  $\leftarrow$  solve(constraint)
        apply(vector,  $M$ )
        sequence  $\leftarrow$  sequence  $\cdot$  {vector}
      else
        v  $\leftarrow$  tempContext
        sequence  $\leftarrow$  tempSequence
        loop  $\leftarrow$  null
        break
      end
    if loop  $\neq$  null  $\wedge \gamma_{t|reg}(v)$  then
      return loop
    end
  end // for loop
end // while hasNext
end // while  $\neg \gamma$ 
return null

```

The pseudo-code utilizes three special functions: *init*(M, s, r) constructs all possible elementary loops in the M 's state graph that start from the state s and include transitions dependent via the register r and returns the iterator that combines a *bounded* number of elementary loops into complex ones (the elementary loops are constructed by using Dijkstra's algorithm to connect dependent transitions); *hasNext*(i) checks whether the iterator i can produce more loops; *next*(i) returns the next loop and updates the iterator i . Note that the limit on the loop length is chosen individually for each design.

IV. THE FATE+ ALGORITHM

To get rid of evident bottlenecks, we have implemented a slightly modified version of the original FATE algorithm, so-called FATE+. Let us consider the changes having been made.

A. Transition Selection

In FATE+'s random traversal, *choose*(T), where T is a non-empty set of transitions, works a bit differently. If there exist uncovered transitions, the function randomly chooses one of them; otherwise, it returns an arbitrary item of T . This minor change significantly increases the effectiveness of the random generation phase.

B. Symbolic Execution

FATE implements an approximate method for checking whether a given path is feasible (**for** $p \in \text{path}$ **do** ... **end**). Let P be a path, t be the last transition of P , r be a register used in γ_t , and v be a context. Given a transition p of P , the algorithm checks whether d defines r . If it does, the following constraint is constructed and tried to be satisfied: $\gamma \leftarrow \gamma_p \wedge \gamma_{t|r}[\delta_p]$. It is worth reminding that $\gamma_{t|r}$ is the minimal conjunctive member of γ_t that includes all occurrences of r , while $\gamma_{t|r}[\delta_p]$ is the formula

produced from $\gamma_{t|r}$ by applying the substitution corresponding to the action δ_p . The method looks inadequate in the sense that if γ is unsatisfiable for some p , it does not really mean that P is infeasible.

We suggest replacing the approximate approach with full-scale symbolic execution that takes into consideration all the variables defined and used along the path. To be more precise, we suggest using the well-known method for computing the *weakest precondition* of a loop-free program, i.e. a sequence of guarded actions, with respect to a postcondition [11]. The main idea is as follows. Let $\gamma \equiv \text{true}$. Starting from the end of P , for each transition p , including t , the following transformation of γ is performed: $\gamma \leftarrow \gamma_p \wedge \gamma[\delta_p]$. Note that the input variables are renamed in such a way that each transition refers to a unique copy of the inputs. As soon as P is processed, all occurrences of the registers are replaced by the values taken from v : $\gamma \leftarrow \text{refine}(\gamma, v)$. P is feasible if and only if γ is satisfiable. A test sequence can be constructed by solving the constraint.

C. Test Reduction

In FATE, there is a frequent situation where multiple test vectors cover the same transition. To overcome the issue, we have introduced a simple test reduction technique. While generating tests, each test sequence is associated with the transitions having been covered. At the end of the process, the set of test sequences W and the set of covered transitions T_{cov} are available. The technique is as follows. First, the transitions reached by unique test sequences are identified. Each test sequence that covers at least one such transition is moved from W to the reduced test R ; all transitions covered by the sequence are excluded from T_{cov} . Then, while T_{cov} is not empty, the following actions are performed. The test sequences that cover largest subsets of T_{cov} are determined; among them, a shortest one is chosen. The selected sequence is moved from W to R , while the covered transitions are removed from T_{cov} .

V. THE RETGA ALGORITHM

The algorithm proposed in this paper is called RETGA (Retrascope EFSM-based Test Generation Algorithm). It has the same phases as FATE; moreover, the EFSM analysis phase is absolutely identical to FATE's one. As FATE+, it uses the modified *choose*(T) function and applies the test reduction. Let us consider the main phases in more detail.

A. Random Traversal

As in FATE, the EFSM models are processed one-by-one; though a different arbitration principle is used. The priority of a model depends on the coverage having been achieved: the better the coverage is, the less the priority is. Such a strategy is to avoid a situation when a covered EFSM of the highest priority prevents generating inputs for poorly covered models.

The pseudo-code for the random traversal is as follows (as before, $\{M_i = \langle S_i, V, T_i \rangle\}_{i=1}^m$ are the EFSMs being tested; *result* is the generated test):

```

result  $\leftarrow$   $\emptyset$ 
coverage  $\leftarrow$   $\emptyset$ 
ignored  $\leftarrow$  0
 $L \leftarrow (\sum_i |T_i|) / (\sum_i |S_i|)$ 
while ignored  $\leq L \wedge$  coverage  $\neq \cup_i T_i$  do

```

```

reset( $\{M_i\}$ )
sequence  $\leftarrow \emptyset$ 
usefulSequence  $\leftarrow \text{false}$ 
transitions  $\leftarrow \emptyset$ 
buffer  $\leftarrow \emptyset$ 
while  $|buffer| \leq L$  do
  vector  $\leftarrow \emptyset$ 
  usefulVector  $\leftarrow \text{false}$ 
  for  $i \in \{1, \dots, m\}$  do
    out  $\leftarrow \{t \in T_i \mid s_t = s_i\}$ 
    while out  $\neq \emptyset$  do
       $t \leftarrow \text{choose}(out)$ 
      out  $\leftarrow out \setminus \{t\}$ 
      constraint  $\leftarrow \text{refine}(\gamma_t, \text{vector} \cup v)$ 
      if isSAT(constraint) then
        vector  $\leftarrow \text{vector} \cup \text{solve}(\text{constraint})$ 
        if  $t \notin \text{coverage}$  then
          usefulSequence  $\leftarrow \text{true}$ 
          coverage  $\leftarrow \text{coverage} \cup \{t\}$ 
        end
        if  $t \notin \text{transitions}$  then
          usefulVector  $\leftarrow \text{true}$ 
          transitions  $\leftarrow \text{transitions} \cup \{t\}$ 
        end
      break
    end
  end // while out
end // for i
apply(vector,  $\{M_i\}$ )
buffer  $\leftarrow \text{buffer} \cdot \{\text{vector}\}$ 
if usefulVector then
  sequence  $\leftarrow \text{sequence} \cdot \text{buffer}$ 
  buffer  $\leftarrow \emptyset$ 
end
end // while sequence
if usefulSequence then
  result  $\leftarrow \text{result} \cup \{\text{sequence}\}$ 
else
  ignored  $\leftarrow \text{ignored} + 1$ 
end
end // while result

```

B. Directed traverse

Before describing the directed traversal phase, let us give some definitions. A *piecewise path* is a sequence of paths, so-called *pieces*, for which there is a path including all of the pieces (with no overlaps) in the given order. Given a register r , a *partial definition path* is a piecewise path that *propagates* at least one input to r and has no transitions not taking part in the propagation. The value of r after executing a path for the given piecewise path depends on at least one input captured in the beginning of the execution.

The directed traversal is performed separately for each EFSM. Here is the pseudo-code (M is the EFSM being tested; *result* is the generated test):

```

targets  $\leftarrow \{t \in (T_M \setminus \text{coverage}) \mid \text{reach}(M, s_t) \neq \emptyset\}$ 
while targets  $\neq \emptyset$  do
   $t \leftarrow \text{choose}(\text{targets})$ 
  path  $\leftarrow \text{shortestPath}^*(M, s_t)$ 
  path  $\leftarrow \text{path} \cdot \{t\}$ 
  if isFeasible( $M, \text{path}$ ) then
    sequence  $\leftarrow \text{solve}(M, \text{path})$ 
    result  $\leftarrow \text{result} \cup \{\text{sequence}\}$ 

```

```

coverage  $\leftarrow \text{coverage} \cup \{t\}$ 
else
  if  $\neg \text{process}(M, t)$  then
    warning "The transition  $t$  cannot be reached"
  end
end
targets  $\leftarrow (\text{targets} \setminus \{t\}) \cup \{t' \in T_M \mid s_{t'} = s_t\}$ 
end // while targets

```

Here, $\text{shortestPath}^*(M, s)$ returns a shortest (in terms of the number of transitions) path from the initial state of the model M to the state s ; $\text{isFeasible}(M, P)$ constructs the weakest precondition of the path P with respect to *true* and checks whether it is satisfiable in the initial context of the model M ; $\text{solve}(M, P)$ satisfies the constraint and converts the solution to the test sequence (uninitialized inputs are randomized). The $\text{process}(M, t)$ function looks as follows:

```

for counter  $\in \{r \in R_M \cap \text{Use}_{\gamma_t} \mid \text{isCounter}(r)\}$  do
  loops  $\leftarrow \{\{\{t_i\}\}_i \mid \{t_i\}_i \in \text{dataDepLoops}(M, \text{counter})\}$ 
  if processLoops( $M, t, \text{counter}, \text{loops}$ ) then
    return true
  end
end // for counter
for define  $\in \text{partialDefPaths}(M, R_M \cap \text{Use}_{\gamma_t})$  do
  if processPieces( $M, t, \text{define}$ ) then
    return true
  end
end // for define
return false

```

In the pseudo-code above, $\text{dataDepLoops}(M, c)$ denotes the set of data dependency loops for the counter c of the model M (each loop starts with the transition that *defines* the counter). As you can see, *loops* is the set of piecewise paths relating to the data dependency loops. $\text{partialDefPaths}(M, R)$ returns the set of partial definition paths for M 's registers of the set R . Here is the description of $\text{processLoops}(M, t, \text{counter}, \text{loops})$:

```

groups  $\leftarrow \text{groupLoops}(\text{loops})$ 
for group  $\in \text{groups}$  do
  loopIterator  $\leftarrow \text{init}(M, \text{group})$ 
  while hasNext(loopIterator) do
    loop  $\leftarrow \text{next}(\text{loopIterator})$ 
    if processPieces(loop  $\cdot \{\{t\}\}$ ) then
      return true
    end
  end // while hasNext
end // for group
return false

```

Here, $\text{groupLoops}(L)$ splits the set of loops (piecewise paths) L into disjoint subsets according to the first transition (which defines the counter). The loop iteration scheme is similar to FATE's one, though each result is a piecewise path. The pseudo-code for $\text{processPieces}(M, t, \{P_i\}_{i=1}^k)$ is shown below:

```

if reach( $M, s_t$ )  $= \emptyset$  then
  return false
end
path  $\leftarrow \text{shortestPath}^*(M, \text{start}(P_1))$ 
for  $i \in \{1, \dots, k-1\}$  do
  path  $\leftarrow \text{path} \cdot P_i$ 
  if  $\neg \text{isFeasible}(M, \text{path})$  then
    return false
  end

```

```

path' ← path · shortestPath(M, end(Pi), start(Pi+1))
failed ← true
if isFeasible(M, path') then
  path ← path'
  failed ← false
else
  for bridge ∈ paths(M, end(Pi), start(Pi+1)) do
    path' ← path · bridge
    if isFeasible(M, path') then
      path ← path'
      failed ← false
      break;
    end
  end // for bridge
end // if isSAT
if failed then
  return false
end
end // for i
path ← path · Pk
if ¬isFeasible(M, path) then
  return false
end
sequence ← solve(M, path)
result ← result ∪ {sequence}
coverage ← coverage ∪ {i}
return true

```

In the pseudo-code, $start(P)$ and $end(P)$ return respectively the initial and the final state of the piecewise path P ; $paths(M, s, s')$ returns the list of cycle-free paths between M 's states s and s' sorted by length.

C. Examples

Let us consider how the RETGA algorithm works on the example of two models, namely EFSM-1 and EFSM-2. Both models correspond to the cases that are difficult for FATE.

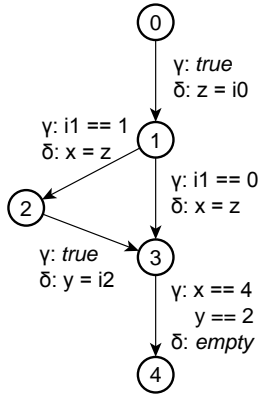


Fig. 1. EFSM-1

In EFSM-1 (see Fig. 1), the random traversal is unlikely to cover the transition $3 \rightarrow 4$ as it requires, first, walking through the path $0 \rightarrow 1 \rightarrow 2 \rightarrow 3$ and, second, assigning $i0 \leftarrow 4$ (while traversing $0 \rightarrow 1$) and $i2 \leftarrow 2$ (while traversing $2 \rightarrow 3$). As for the directed traversal of $3 \rightarrow 4$, the following partial definition paths are found for the registers x and y used in the transition's guard:

1. $0 \rightarrow 1 \rightarrow 3$ ($i0$ is propagated to x via z).
2. $0 \rightarrow 1 \rightarrow 2$ ($i0$ is propagated to x via z).

3. $2 \rightarrow 3$ ($i2$ is directly assigned to y).

The first path does not initialize y and has no continuations that could do that. For the second one, the pieces $\{0 \rightarrow 1 \rightarrow 2, 3 \rightarrow 4\}$ are composed and supplemented by the only “bridge” $2 \rightarrow 3$. For the third path, the “prefix” $0 \rightarrow 1 \rightarrow 2$ explored at the random traversal phase is put before the partial definition path. In both cases, the path $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ is constructed. To check whether the path is feasible, the weakest precondition is computed: $i0[1] = 4 \wedge i1[2] = 1 \wedge i2[3] = 2$ (the indices in the square brackets refer to the positions of the test vectors in the test sequence). It is satisfiable; the solution is as follows:

1. $i0 = 4$; $i1$ and $i2$ are randomly valued;
2. $i1 = 1$; $i0$ and $i2$ are randomly valued;
3. $i2 = 2$; $i0$ and $i1$ are randomly valued;
4. $i0, i1$ and $i2$ are randomly valued.

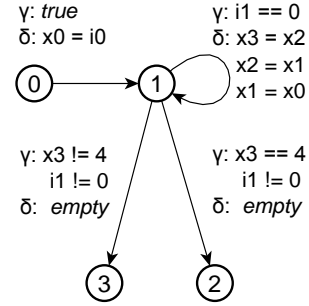


Fig. 2 EFSM-2

In EFSM-2 (see Fig. 2), a transition of the interest is $1 \rightarrow 2$. The shortest path that reaches the transition is $0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 2$ with the assignment $i0 \leftarrow 4$ on the first step. There is only one partial definition path for $x3$, namely $0 \rightarrow 1 \rightarrow 1 \rightarrow 1$. The path can be supplemented only with the target transition, which gives $0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 2$. The weakest precondition is $i0[1] = 4 \wedge i1[2] = 0 \wedge i1[3] = 0 \wedge i1[4] = 0 \wedge i1[5] \neq 0$. It is readily seen that the condition is satisfiable.

VI. EXPERIMENTAL RESULTS

The RETGA algorithm has been implemented as a part of the Retrascope [12] project. It uses the Fortress [14] library together with the Z3 [15] solver for representing expressions and solving constraints. To compare the algorithm with FATE and FATE+, the ITC'99 benchmark [13] was utilized.

Table I shows the characteristics of the EFSMs extracted from some ITC'99's designs. As it has been already said, we used the extended variant of the method described in [8] to build the models, though all of the presented approaches do not depend on the way EFSMs are produced.

TABLE I. CHARACTERISTICS OF THE EXTRACTED EFSMS

Design	Number of States	Number of Transitions
b01	8	24
b02	7	17
b04	3	29
b06	7	33
b07	8	21
b08	4	12
b10	11	38

Table II and Table III show the test generation results. All generators achieve 100% coverage for b01, b02, b04 and b06 and 95% coverage for b07 (there is an infeasible transition). The difference in coverage reached by RETGA and FATE / FATE+ for b08 is due to the fact that FATE and FATE+ handle data dependencies in a simpler way; in particular, they do not try different “bridges”. The difference in coverage reached by FATE and FATE+ for b08 and b10 demonstrates the advantage of the symbolic execution over the simplified approach used in FATE. The difference in size of the tests generated by FATE and FATE+ relates to the test reduction technique applied in FATE+. The RETGA’s tests are usually shorter since it rejects redundant random vectors.

It is significant to note that the L and N parameters (which are related to the random traversal phase of FATE and FATE+) were set to $\sum_{i=1}^m |S_i|$ and $\sum_{i=1}^m |T_i| / \sum_{i=1}^m |S_i|$ respectively. The loop iteration limit (which is relevant for all of the generators) was set to 8 (this value is enough for b07 and b08, whereas other designs have no counters).

TABLE II. NUMBER OF TEST VECTORS IN THE TESTS

	FATE	FATE+	RETGA
b01	115	70	49
b02	62	48	33
b04	104	104	36
b06	198	100	76
b07	246	208	166
b08	31	31	52
b10	173	170	135

TABLE III. TRANSITION COVERAGE ACHIVED BY THE TESTS

	FATE	FATE+	RETGA
b01	100%	100%	100%
b02	100%	100%	100%
b04	100%	100%	100%
b06	100%	100%	100%
b07	95%	95%	95%
b08	75%	83%	100%
b10	89%	100%	100%

The tests generated by RETGA were applied to the designs by using the Questa simulator [16]. The source code coverage having been achieved is presented in Table IV (each column corresponds to some metric of the Questa coverage report). It can be seen that the code coverage rather is high.

TABLE IV. SOURCE CODE COVERAGE REACHED BY RETGA

	Statements	Branches	FSM States	FSM Transitions
b01	100%	100%	100%	100%
b02	100%	100%	100%	100%
b04	100%	100%	100%	100%
b06	100%	100%	100%	100%
b07	93.93%	94.73%	100%	100%
b08	100%	100%	100%	100%
b10	100%	100%	100%	100%

VII. CONCLUSION

In this paper, an EFSM-based test generation algorithm has been proposed. The approach allows reaching better transition coverage with less number of test vectors than the known methods. However, the research is still in progress; there are many issues to be solved. Let us mention some of them. First, the approach is hardly applicable to complex hardware designs involving a great number of tightly connected EFSMs. It uses a simple coverage-based heuristic to decide which EFSM to handle next, whereas advanced techniques are expected to rely on the semantics of a system under test. Second, the method for searching “bridges” needs to be optimized. Being irrelevant for simple EFSMs (as ones presented in Section VI), this issue is of high value and importance for real-life hardware. Third, in the current implementation, each guard (each constraint, in general) is viewed as an indivisible entity and solved as a whole. It is not an issue as long as the goal is to cover EFSM transitions, but it may lead to poor expression coverage as there are many ways to satisfy a constraint. Finally, the quality of testing strongly depends on the models being used, while the EFSM extraction process is ambiguous (in the trivial case, a single-state EFSM is constructed for an arbitrary design). It seems to be useful to formalize a notion of a “good” model.

REFERENCES

- [1] J. Bergeron, “Writing Testbenches: Functional Verification of HDL Models”, *Kluwer Academic Pub*, 2003.
- [2] J. Blyler, “Are Best Practices Resulting in a Verification Gap?”. <http://chipdesignmag.com/sld/blog/2014/03/04/are-best-practices-resulting-in-a-verification-gap/>
- [3] V. Jusas, T. Neverdauskas, “FSM Based Functional Test Generation Framework for VHDL”, *Proceedings of International Conference on Information and Software Technologies (ICIST)*, 2012, pp. 138-148.
- [4] A.Y. Duale, M.U. Uyar, “A Method Enabling Feasible Conformance Functional Test Sequence Generation for EFSM Models”, *IEEE Transactions on Computers*, 53(5), 2004, pp. 614-627.
- [5] V.G. Lazarev, E.I. Pijl, “Sintez upravljajushhih avtomatov”, *Energoatomizdat*, Moscow, 1989, 328 p. (in Russian)
- [6] K.T. Cheng, A.S. Krishnakumar, “Automatic Generation of Functional Vectors Using the Extended Finite State Machine Model”, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1996, pp. 57-79.
- [7] G. Di Guglielmo, L. Di Guglielmo, F. Fummi, G. Pravadelli, “Efficient generation of stimuli for functional verification by backjumping across extended FSMs”, *Journal of Electronic Functional testing: Theory and Application*, Volume 27, Issue 2, 2011, pp. 137-162.
- [8] A. Kamkin, S. Smolov, “The method of EFSM extraction from HDL: application to functional verification”, *Proceedings of the conference on Problems of Perspective Micro- and Nanoelectronic Systems Development*, Part II, 2014, pp. 113-118.
- [9] Z. Navabi, “Languages for Design and Implementation of Hardware”, W.-K. Chen (Ed.). *The VLSI Handbook*. *CRC Press*, 2007. 2320 p.
- [10] E.W. Dijkstra, “A note on two problems in connexion with graphs”, *Numerische Mathematik*, 1, 1959, pp. 269-271.
- [11] E.W. Dijkstra, “A Discipline of Programming”, *Prentice Hall*, 1976, 217 p.
- [12] Retrascope toolkit. <http://forge.ispras.ru/projects/retrascope>
- [13] ITC’99 benchmark. <http://www.cad.polito.it/tools/itc99.html>
- [14] Fortress library. <http://forge.ispras.ru/projects/solver-api>
- [15] Z3 solver. <http://z3.codeplex.com>
- [16] Questa simulator. <http://www.mentor.com/products/fv/questa/>

Remote Service of System Calls in Microkernel Hypervisor

Kurbanmagomed Mallachiev

Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russian Federation
mallachiev@ispras.ru

Nikolay Pakulin

Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russian Federation
npak@ispras.ru

Abstract—This paper presents further development of Sevigator hypervisor-based security system. Original design of Sevigator confines users' applications in a separate virtual machine that has no network interfaces. For trusted applications Sevigator intercepts network-related system calls and routes them to the dedicated virtual machine that services those calls. This design allows Sevigator protect networking from malicious applications including high-level intruders residing in the kernel.

New hypervisors built on top of microkernel opened the door to redesign of Sevigator. Those hypervisors are small operating systems by nature, where most of hardware operations as well as management of virtual machines are isolated in processes with low priority level. Compromising such a process does not result in compromising the whole hypervisor.

In this paper we present the results of experiment with NOVA hypervisor where system calls of trusted applications are serviced by a dedicated process in the hypervisor rather than a separate VM. The experiment shows about 25% performance gain due to reduced number of context switches.

Keywords—virtualization hypervisor security microkernel

I. INTRODUCTION

The main purpose of the project is to develop a security facility that protects data confidentiality on a computer connected to the Internet and managed by an untrusted operating system. We assume that malicious code can get unlimited access to all hardware and software system resources through vulnerability or backdoors in system software.

Today popular modern operating systems (such as Linux or Windows) are based on monolithic kernel, where all components of kernel have equally high privileges. In this case if malicious code penetrates OS kernel, then there is a risk of losing control over any OS resources including application in-memory data, confidential information in file storage, etc. Integrity and confidentiality of data transmitted over the network are also threatened, even in the case when cryptography is used.

The question is whether it is possible to protect unmodified applications that run under unmodified commodity OS like Windows or Linux on a commodity workstation with x86 CPU. Protection systems located in kernel, such as antivirus,

firewall, intrusion detection, can themselves be attacked by privileged malicious code. Possible way of protection from those attacks is the transfer of protection to more privileged level.

The answer is “probably yes”: a prototype called Sevigator [2, 3, 4, 5] protects applications in Linux from malware and comprised kernel. It uses hardware-assisted virtualization [1] to secure operating memory of applications and control access to communication hardware (network interface card). It allows to launch OS under control of virtual machine monitor (VMM, also called hypervisor). Hypervisor is much smaller than OS, fully isolated from it, and has higher privilege than OS. Hardware virtualization is supported by most modern processors, which suggests the possibility of widespread use of security systems based on hypervisors

Sevigator provides isolation of untrusted OS from network, but keeps operability of trusted application. For them, and only for them, an access to network resources is granted. An important feature of this approach is that there is no need to modify or recompile any applications or OS

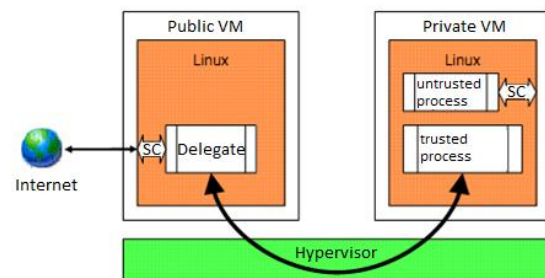


Fig. 1. Sevigator architecture

Within Sevigator approach OS resides in a virtual machine, while protection system is located in hypervisor. It provides facilities to isolate untrusted applications from network access; to prevent data leaks due to code intrusion or memory attacks it controls memory integrity of the applications under protection. The hypervisor provides simultaneous execution of two completely isolated from each other virtual machines. First called *private* is primary one, user interacts with it, and it believes that network adapter is physically absent. The second VM called *public* is service system which has unlimited access

to network. Network support for trusted processes in private machine is provided by hypervisor through remote execution of required (limited) set of system calls in the public virtual machine. Full description of security algorithms can be found in [2, 3, 4, 5].

We refer to this scheme as *remote servicing of system calls* since the hypervisor intercepts parameters of a system call in the private VM and transfer them to the public VM, where the actual code is executed.

The scheme with two VMs was motivated by the following considerations: isolation networking operations from private machine and minimization the risk of hypervisor compromise in the case of compromised network component. Isolation makes network access possible only for trusted application. Execution within public VM means that compromise of the VM will not lead to compromise of hypervisor kernel.

Sevigator system originally was based on hypervisor KVM (Kernel-based Virtual Machine), and using the second VM was the only possible solution to satisfy the constraints. Later Sevigator without changes of its architecture was ported to NOVA microkernel hypervisor [6].

Our work shows that using hypervisor based on the microkernel architecture allows us to replace the second virtual machine with a process in hypervisor with same functionality. This is possible because microkernel isolates processes and executes them at lower privilege level than the kernel. And this change significantly reduces overhead of having dedicated OS only for remote execution of service calls.

II. HYPERVISORS OVERVIEW

There is a lot of hypervisors and they use different ideas. We chose NOVA [7] to port Sevigator because it was the only one that satisfied own requirements for original Sevigator design (requirements and hypervisor comparison can be found in own work [6]). And when we ported Sevigator, NOVA architecture gave us idea how we can redesign Sevigator to reduce overhead but keeping security.

With new design of Sevigator, where dedicated process is responsible for servicing system call, we again looked if it can be implemented in different hypervisor besides NOVA. The following hypervisors were considered: BitVisor[8], SecVisor[9], Xen[10], Qubes OS [11]. All of them are distributed under open source licenses and don't require existence of a host operating system.

BitVisor is hypervisor and virtual machine monitor (VMM), designed to ensure security of computer systems. BitVisor provides encryption of network connections and data on disk. Ensuring confidentiality of network and disk data is transparent to the operating system. BitVisor designed to create minimal overhead on encryption and decryption of data.

Bitvisor doesn't separate VMM and kernel of the hypervisor, so performed at the same privilege level. BitVisor supports exactly one virtual machine - this is done in order to minimize the overhead on the interaction of the guest OS with the devices, primarily input and output devices. Bitvisor based

on paravirtual architecture: hypervisor intercepted memory access and I/O access, and pass-through anything else. Bitvisor intercept accesses to protect hypervisors from the guest OS, and enforce security functionalities. Bitvisor cannot execute processes at lower privilege level. So Bitvisor didn't satisfy us.

SecVisor is a very small hypervisor (about 10 times smaller than NOVA) which goal is protecting OS kernel against an attacker who controls everything but the CPU, the memory controller, and system memory chips.

SecVisor provides a lifetime guarantee of the integrity of the code executing with kernel privilege. In other words, SecVisor prevents an attacker from either modifying existing code in a kernel or from executing injected code with kernel privilege, over the lifetime of the system. SecVisor ensures that only code approved by the user can execute with kernel privilege. SecVisor also execute all its parts at the same privilege level. SecVisor was rejected.

Xen is a very popular virtualization platform, which is widely used to build cloud services. Xen virtualization platform includes a hypervisor, virtual machine monitor for guest OS, dedicated virtual machine dom0 to work with devices and specialized drivers to access the device via the dom0. These drivers are called paravirtualized as they "know" that the OS is running under Xen and effectively interact with the hypervisor and dom0.

Xen hypervisor implements the minimum set of operations: management of RAM, processor status, real time clock, interrupt processing and control of DMA (IOMMU). All other functions, such as the implementation of virtual devices, creation and deletion virtual machines, moving VMs between servers in the cloud, etc. is implemented in a dedicated virtual machine dom0.

All functions related to ensuring network performance, disk drives, video cards emulation and other devices placed outside the hypervisor. Typically, the request handling devices consist of two parts. Driver in the guest operating system translates requests from the OS to program handler in dom0. To increase the security of the system servers, virtualize devices run as separate processes in OS dom0. Failure in such a program leads to a denial of only one virtual device in one VM and does not affect the work of other copies of the server.

Xen architecture requires using dedicated virtual machine for servicing network-related system calls and this is a big overhead. Xen didn't satisfy us

Qubes OS is a hypervisor based on Xen. Qubes implements a security by isolation approach. In Qubes, the isolation is provided in two dimensions: hardware (separated network domain, storage domain, GUI) and software (domain with different levels of trust e.g. work domain – most trusted, shopping domain, random domain – less trusted). Domains separated by executing at different virtual machines. So this system also has same disadvantage as Xen, and was rejected too.

III. ORIGINAL SEVIGATOR DESIGN

A. General Architecture

Among the applications running in the OS, the protection system identifies several applications that are considered as trusted. All others applications are considered as untrusted. The security problem is to prevent the leakage or compromising of confidential data of trusted applications. Trusted applications for the normal functioning may require access to the public network. This network connection can be used by malicious code in the OS kernel for the leakage of sensitive data.

The solution is based on use of hardware virtualization technology, execution of an OS in the virtual machine (VM), and implementation protection system in the body of a virtual machine monitor (hypervisor) [2]. The hypervisor provides simultaneous execution of two completely isolated from each other virtual machines (fig. 1). Both are running the same untrusted OS. The first VM, we will call it *private*, is the primary one. It is there where critical data resides and applications (both trusted and untrusted) are executed processing those data. Hypervisor blocks access to the network interface for private VM and its guest OS believes that the network adapter is physically absent. Thus, even if malicious code managed to gain access to critical data, it will not be able to transfer them to the outer world.

Network access for trusted applications is supplied by the second VM called *public*. It has free access to the network. However, due to VMs isolation provided by the hypervisor the software in the public VM (including OS kernel) cannot gain access to data residing within the private VM.

Network support for trusted processes is implemented through remote servicing of required set of system calls in the public VM. The hypervisor intercepts network-related system calls invoked by a trusted process, analyzes the data and, when necessary, transmits them to the public VM. Note that the remote service of the system call is made transparent for a trusted process and an OS.

B. NOVA based architecture

NOVA is a microkernel for hypervisor. NOVA itself is only a kernel, for running virtual machines you should use one of the environments, built atop of it: NUL, NRE or Genode. We use NUL because NRE still misses some NUL features, and Genode is much larger.

Because of microkernel design, only the NOVA kernel runs with the highest priority and every process of NUL is executed as user space process with priority level CPL3 (lowest on Intel IA-32 architecture).

NUL is an experimental operational environment and it is still work in progress. It contains a number of simplified components, e.g. direct access to host PCI devices works unstable. As a result VMM (Virtual Machine Monitor) has to emulate hardware devices for the guest virtual machine. And if the emulated model needs access to a host device, than you need to have driver in NUL for that device. For networking NUL provides a small number of drivers, most notable is the classic NE2000 network card. For our experiment we used

NE2000-compatible network card RTL8029AS, for which NUL has a driver.

The port of Sevigator architecture to NOVA hypervisor uses two virtual machines [13] to service network-related system calls of trusted users' applications. As an example Fig. 2 shows how servicing *send* system call works.

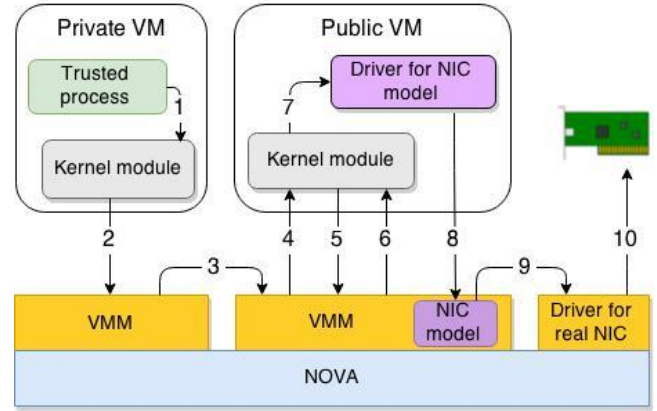


Fig. 2. Path of send message in original design Sevigator

Yellow colored boxes are processes in NOVA. Interaction with and between processes always imply calling NOVA kernel, but for simplicity we don't show them on the figure.

When trusted process executes *send* system call the Sevigator module in OS kernel intercepts it (1), forms special fixed size message and free size vault and execute hypercall (2). VMM passes (3) the message and the vault to another VMM. This VMM sends (4) the message to public VM kernel module. Module finds vault size in message, allocates memory, asks (5) for vault and receives (6) it. Module forms a new message and sends it to Linux kernel, which calls (7) network driver for network card emulated by VMM. The driver sends (8) bytes to the network card model, which passes (9) them to driver of the actual card. And finally the driver in the hypervisor sends bytes to the network card.

As we can see the path that passes network messages is really long. In the next chapter we will show how to achieve a shorter path.

IV. NEW SEVIGATOR DESIGN

Microkernel based hypervisor allows us to redesign Sevigator. Those hypervisors have well isolated parts. Only a small kernel has highest priority level. Most of hardware operations as well as management of virtual machines are isolated in processes with low priority level.

The idea of the redesign is to move servicing system calls of trusted applications to hypervisor applications. Having dedicated processes in hypervisor we keep all pluses of using dedicated virtual machine such as isolation of servicing system calls in code and securing the risk of compromise the system by reduction of priority level. It means that compromising such a code doesn't mean compromising the whole hypervisor. But redesigning gives more: it reduces trusted code base from millions of lines of code (LoC) for

public VM to tens of thousands LoC for dedicated applications in hypervisor. And also we reduce overhead of context switching: redesigned system don't need at least context switching between VMM and public VM; so we increase performance of the whole system.

In our paper we present a proof of concept of the new approach to servicing system calls of trusted applications in dedicated environment.

We selected networking system calls for study. Fig. 3 presents the idea: networked system calls are serviced in the dedicated process over NOVA microkernel. The application is based on popular embedded TCP/IP stack called lwIP[12]. The application is a wrapper around lwIP that parses the parameters of remote system calls and invokes corresponding lwIP operations. In the following text we will refer to this application as "lwIP".

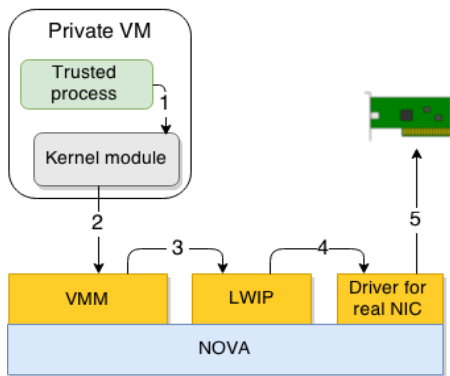


Fig. 3. Path of send message in redesigned Sevigator

Fig.3 shows servicing of send message in redesigned system. Here we will only discuss difference of redesigned system. Steps (1) and (2) are same as in original design. VMM sends (3) message and vault to LwIP process, which analyses message, understands what system call was called, and forms a packet, that will be sent (4) to driver. Driver sends bytes to the real network card.

We can see that in new design path is much shorter, so new design should work faster. How fast you can see in next chapter. In order to support the concept of socket used by trusted application we implemented a small glue layer over lwIP. The prototype implementation support socket *create* and *close*, socket *bind* and *connect*, *send* and *recv* for TCP and UDP. Raw sockets (e.g. ICMP messages) are not supported yet.

V. PERFORMANCE

We conducted an experiment to measure network performance of the redesigned system. During experiment we compared performance of the original design with two VMs, and the new design with the dedicated process. As the reference point we used native Linux running on hardware without hypervisor and ran hypervisor with pure lwIP application without VMM.

All measurements were performed on the same machine with AMD Phenom II x4 980 3.7 GHz CPU, 16 GB RAM. As network card we used once popular RTL8029AS card. It is ne2000 compatible and is one of the few cards supported by NOVA/NUL. The card is 10Mbit/s. We use this old card because other cards supported by NOVA turned out to be much harder to find.

For testing we run test application in Linux, which executes 1000 times *sendto* system call, sending UDP packets to the network. We were sending short 60 bytes message. The destination workstation was receiving the packets, identified lost packets and measured time between the first and the last packets. We didn't measure time at the guest virtual machine because return from *sendto* call doesn't mean that packet was sent.

Fig.4 shows the test performance difference among original and new architectures and native Linux.

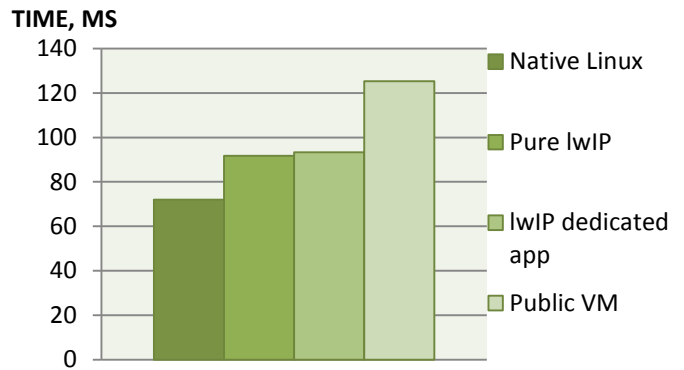


Fig. 4. Time for sending 1000 UDP packets

The experiments showed that replacing the virtual machine with a dedicated application increased performance by 26%. The overhead compared to the native Linux execution was reduced from almost 100% to 29%.

Comparing with pure lwIP case shows that current overhead for transfer system call in lwIP is only 1.4 μ s. For 10 Mbit/s network this is insensitive. The bottleneck of current realization is lwIP and NE2000 driver. The NE2000 driver in Nova is far from perfection and careful queuing of pending packets may reduce the total overhead even more.

Servicing of system calls in an application compared to a dedicated VM simplifies the flow control. Removing the second VM resulted in omitting:

1. interrupt injection in the public VM (required to notify the VM that there are packets pending);
2. VM exit to pass frames from public OS to NIC model in the VMM;
3. IPC calls between VMM and NIC driver in the hypervisor.

Another important gain is significant reduction of the trusted code base required for servicing network-related system calls. The design with two virtual machines implied

that we have to trust the whole Linux kernel, e.g. millions lines of code due to the minolitic nature of that kernel. When the system calls are serviced by the lwIP application, the trusted computing base shrinks to about 70 000 LoC, the size of lwIP.

VI. FUTURE WORK

In future we want to develop NUL drivers for modern network cards and make experiments on them. Also because NOVA UserLand was made as a test project and isn't fully stable for now, we have encountered problems with memory management, and have errors while working with big packets. We want to find the causes the revealed problems and fix it.

And finally we will port guest modules to modern Linux kernel and see if there are any changes in performance.

VII. CONCLUSION

Our work shows that using microkernel based hypervisors open new perspectives and creates new approach to servicing hardware requests from guest OS in hypervisor.

Using microkernel hypervisor allow us to redesign system by moving system call servicing in hypervisor application. Those hypervisors executes process with low priority, so compromising of application doesn't lead to compromising of the whole hypervisor.

We were able to move servicing of system calls to such a process. It significantly reduces overhead for servicing network-assisted system calls and speeds up execution: new design makes network connection 30% faster. Furthermore it reduced trusted code base by two orders of magnitude, and this is very important for security system, because it makes audit or verification of system simpler.

REFERENCES

- [1] Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, and 3C: System Programming Guide [Online]. Available: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-system-programming-manual-325384.pdf>
- [2] I. Burdonov, A. Kosachev, P. Iakovenko Virtualization-based separation of privilege: working with sensitive data in untrusted environment. //1st Eurosyst Workshop on Virtualization Technology for Dependable Systems, New York, NY, USA, ACM. 2009. P. 1-6.
- [3] D.V. Silakov. Using Hardware-assisted Virtualization in the Information Security Area. pp. 25-36. Proceedings of the Institute for System Programming of RAS, volume 20, 2011. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)
- [4] P. Iakovenko. Transparent mechanism for remote system call execution. pp. 221-242. Proceedings of the Institute for System Programming of RAS, volume 18, 2010. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)
- [5] P. Iakovenko. Ensuring confidentiality of information processed on a computer with a network connection. Information security problems. Computer Systems. №4. 2009. pp. 23-41. (In russian)
- [6] K. Mallachiev, N. Pakulin. Protecting Applications from Highly Privileged Malware Using Bare-metal Hypervisor. DOI: 10.15514/SYRBOSE-2014-8-10.
- [7] U. Steinberg and B. Kauer. 2010. NOVA: a microhypervisor-based secure virtualization architecture. In Proceedings of the 5th European conference on Computer systems (EuroSys '10). ACM, New York, NY, USA, 209-222.
- [8] T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. Kato. 2009. BitVisor: a thin hypervisor for enforcing i/o device security. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '09). ACM, New York, NY, USA, 121-130.
- [9] A. Seshadri, M., Ning Qu, and A. Perrig. 2007. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. *SIGOPS Oper. Syst. Rev.* 41, 6 (October 2007), 335-350. DOI=10.1145/1323293.1294294 <http://doi.acm.org/10.1145/1323293.1294294>
- [10] C. Takemura and L. S. Crawford. The Book of Xen. No Starch Press. October 2009, 312 pp. ISBN-13 978-1-59327-186-2,
- [11] J. Rutkowska. Software compartmentalization vs. physical separation. Invisible Things Lab, 2014 http://www.invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf
- [12] Adam Dunkels lwIP, a small independent implementation of the TCP/IP protocol suite. <http://www.nongnu.org/lwip>

Constructing Private Service with CRYP2CHAT application

Andry Kiryantsev

Volga Region State University of Telecommunications and
Informatics Moskovskoe sh. 77, Samara, Russia
Email: @ reyzor2142@gmail.com

Iran Stefanova

Volga Region State University of Telecommunications and
Informatics Moskovskoe sh. 77, Samara, Russia
Email: aistvt@mail.ru

Annotation – The paper contains the description of a private service with an encryption and decryption of data on client-side. The focus is on the direct client-to-client connection. The article provides the algorithm for work of the programs. The authors describe the methods of protecting from some network attacks and the experiment of prototype work. Additionally, we consider some potential dangers of an external character that can violate confidential communication data.

Keywords – *cryptography, encryption, encoding, MITM-attack, end2end encryption, node.js, cryptico, javascript*

I. INTRODUCTION

Modern society is characterised by the exchange and buffering information in electronic form. While processing the information, we may need to react immediately on constantly emerging problems with data protection and security of data centres.

The problem is now becoming more urgent considering declarations and current publications by Edward Snowden, the former system administrator for the Central Intelligence Agency. He reports on the fact that the National Security Agency (NSA) operates global surveillance programs with the cooperation of telecommunication companies and European governments through the existing communication networks and systems.

Nowadays to exchange information on-line special programs – messengers – are used. They are particularly useful for transmission of text messages, sound signals, images, video and games as well as for organisation of teleconferences by coding messages of on-line users. Messengers usually operate in coordination with a server, and they are defined as client-side programs with their own rules of work and peculiarities in operating, e.g. ICQ, Skype. The main drawback of these programs is that while using them, we leave metadata on a hosting server as non-encrypted data flow, which provides an opportunity to learn (if required) the information about the common users, time of their communication, the number of messages they send within a session.

II. DESCRIPTION OF CRYP2CHAT PROGRAM

To eliminate the defect we develop a model to run a program that allows coding the data on the client side with the help of Cryp2Chat Application

Currently existing Internet messengers fail to perform the following functions:

- to check for MITM (Men in the middle)-attacks;

- to provide a ‘clean’ (data free) server;
- to destruct messages automatically after the session is over.

MITM-attack is the most wide-spread way to attack for stealing the data of some users. This type of attack presupposes that the attackers are able to read and alter messages of a sender and a receiver as they wish. Additionally, neither a sender nor a receiver sees any hints of the attacker to be in the channel. It is the matter of no importance if SSL cryptographic protocol is applied or not. The attacker hooks into a channel between users and interferes actively with the communication protocol. He/ she may delete, falsify data or provide the false ones.

The term ‘clean’ server implies that the communication between two users leaves no information on the server. In this case the server functions as a repeater and simply translates the encrypted message between the clients. After the session is over, the access to the data of the on-line chat is lost without any opportunity for return.

The described problems with messengers could be solved if we use a new application – Cryp2Chat.

Cryp2Chat application has been developed to minimise the drawbacks of the Internet messengers, i.e. it leaves no metadata on the central server. The client is the only person who can decode the incoming message. The client possesses data de-encryption key, and the key does not go further.

The program operation procedure is the following (fig. 1). A server receives a list of network user’s contacts. A data encryption key is generated on the side of a sender. Further the public key is sent to the server and, finally, to a receiver. The private part of a key remains on the user’s (sender’s) side.

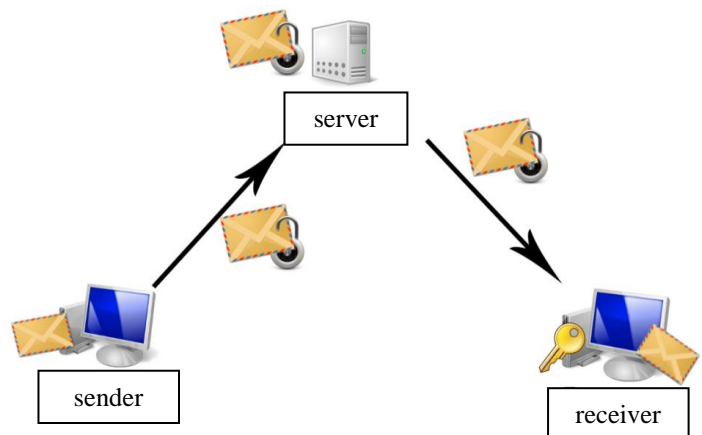


Fig. 1. An Example of Cryp2Chat Application Running

When a user (a receiver) sends back a message, the operation is realized within three main stages:

1. He/she receives a public key of a receiver from the server;
2. The message is encrypted by a public key;
3. The cryptographed message is sent to the server.

RSA method is employed for encryption; the key length includes 1024 bit. However, the possibility to use other algorithms of encryption is also provided.

The server created as a prototype of this application is written in Node.js programming language (advanced

JavaScript) on the basis of Socket.IO library.

Cryp2Chat application is an original service designed to exchange rapidly-changing messages. It supports End2End encryption.

To enable the program to use proxy servers (to protect the client's computer from some network attacks) and to increase the reliability of a channel, we offer the use of network of TOR (The Onion Router). On the computer of a client a proxy server connected to the network of TOR starts its work [1]. It involves a multilevel encryption. The process of message transmission in a network is schematically presented on fig. 2.

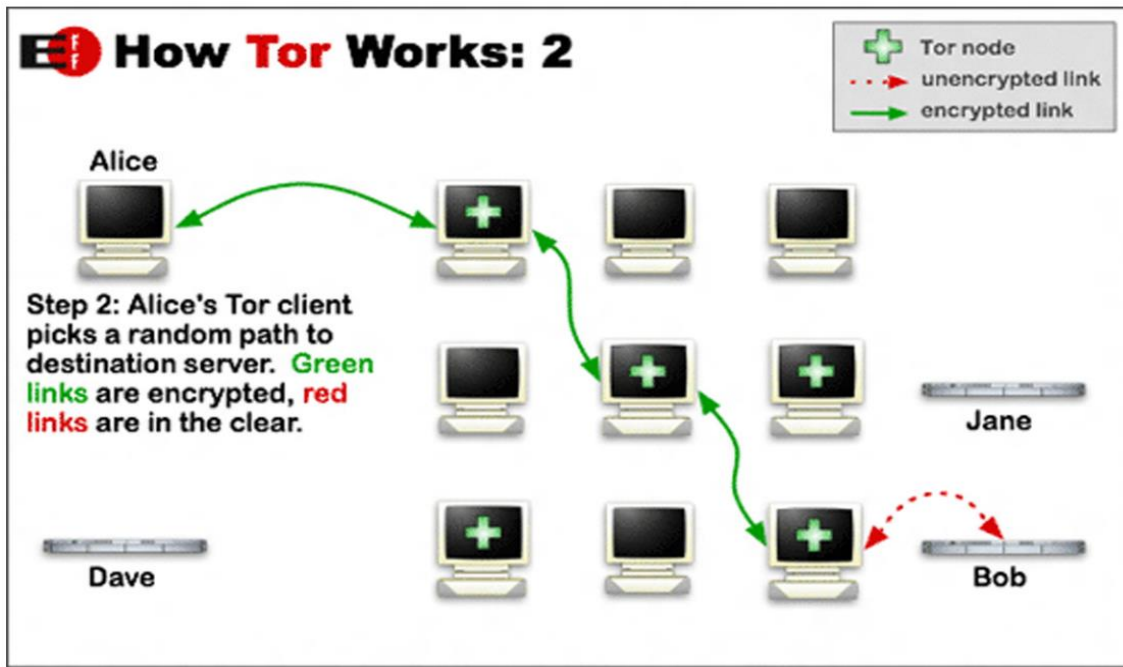


Fig.2. Schematic Presentation of TOR Work.

Before transmitting the data packet to the server, it goes through three random computers. Before being sent, the package is encrypted by three keys: for each of the three computers respectively. In addition, the TOR network can provide anonymity for servers.

Network users start TOR multi-level ("onion") proxy server on their machine. It connects to the TOR servers, periodically forming a chain through the TOR network that uses a multi-level encryption. Every packet entering the system passes through three different proxy servers - server nodes that are randomly selected. Before being sent, the package is sequentially encrypted by three keys: first, in the third node, then in the second node, and, finally, in the first node. When the first node receives a packet, it decrypts the "upper" layer encryption (similar to how we clean the onion) and gets the information where to send the packet to. The second and the third servers do the same. At the same time, the software multi-level ("onion") proxy server provides a SOCKS-interface.

SOCKS (SOCKEt Secure) are the programs, running on the SOCKS-based interface. Their work could be configured through the TOR network. The TOR network creates

multiplexed traffic and sends data through a virtual chain of the TOR network, thus, providing anonymous web surfing.

Inside the TOR network the traffic is forwarded from one router to another, and finally it reaches the exit point from which the pure (unencrypted) data package comes to the original recipient address (server). The traffic from the receiver is sent back to the exit point of the TOR network [3].

The server prototype of this application is written in Node.js (advanced JavaScript) with the help of the library for web sockets - Socket.IO.

Node.js is a programming platform founded on V8 database engine that translates JavaScript into the machine code. In this way it transforms JavaScript from the highly-specialised language into the common language for users. The client part is realized on Html and JavaScript with the help of Crypico library.

Node.js has not been chosen by chance. This is one of the few servers that work quickly and productively with a single-threaded code. For instance, being the programming language it does not need to create a new thread to transmit a stream of query parameters and to interpret the code.

Node.js is the aggregate of the V8 database engine used in Google Chrome and in the abstraction to access the file system and similar server modules.

To shift away from the standard web2.0 scheme of data transmission we used Web-Sockets and their implementation for node.js servers in the form of Socket.IO library. It should be mentioned that Web-Socket is a Protocol intended for exchanging messages between the browser and the web server in real time.

At the same time, the Socket.IO library provides a good level of abstraction above the sockets that are implemented in JavaScript. With its help you can easily pass objects to the server and from the server, without serializing them.

The structure of the server part is the following: the server accepts the message. If it is a command, the server performs certain actions. If it is simply a message, the server sends it to the client.

The JavaScript language, which is used in the prototype, is currently the most common cross-platform language. It is commonly used as an embedded language for program access to the application objects. The JavaScript language is widely used in browsers as a scripting language to add interactivity to the web-pages.

The JavaScript language may be distinguished by its main architectural features: dynamic typing, weak typing, automatic memory management, prototype programming, functions as the first class objects.

The only requirement for JavaScript work (and it is present by default in all operating systems) is the availability of the browser. It does not need to be rewritten when migrating from one operating system to another. We write the script and run it in the place where there is a browser on an electronic device.

Over the last decade JavaScript turned from the applied language for checking how the blanks are filled, into a language that can provide the programmer a powerful tool to tackle any kind of problems. The JavaScript library is constantly updated with new scripts and styles.

Now there are many add-in settings for JavaScript as its possibilities are constantly growing, but the syntax and its architecture is not changed. A simple example is CoffeeScript language, which allows you to write more compact code compared to JavaScript. It helps to solve some architectural omissions such as the lack of OOP (object oriented programming), collbecki (Callbacks) – callback and syntactic ‘sugar’ (code lines that improve the way the program looks like). All this makes the language more convenient for the programmer.

III. PROTOTYPE WORK

As an example, we may consider the fragments of scripts in Cryp2Chat prototype. Below there is a fragment of the script that implements the simultaneous exchange of encryption keys between clients:

```
socket.on('key1',function(data)
{
```

```
    keys[0] = data.key;
});
socket.on('key2',function(data){
    keys[1] = data.key;
    chat.emit('key', { key1: keys[1], key2: keys[0],
        stats: "ok"});
});
```

When the client sends his/her first client key ‘key1’, it is immediately saved. However, while sending the second client key ‘key2’, the handshake happens. The handshake process is asynchronous exchange of public keys to encrypt data between two clients.

In Cryp2Chat prototype the transmission of the incoming message is presented through the following scrip:

```
socket.on('msg', function(data)
{
    socket.broadcast.to(socket.room).emit('receive',
{msg: data.msg, user: data.user, img: data.img});
}
);
```

Next, when the server receives an incoming message, the server sends it to the second client with the help of the socket.io library.

The public RSA key is generated in the following lines of script:

```
var myRSAkey = cryptico.generateRSAKey(PassPhrase,
512);
var PublicKeyString =
    cryptico.publicKeyString(myRSAkey);
```

The decryption of the cryptogram and its presentation in the client side is represented by the lines of the script:

```
var msgs = cryptico.decrypt(textarea.val(),roomKey);
socket.emit ('msg',{msg: msgs, user: name, img:
img});
```

The client is the only one who can decrypt the transmitted message, as the private key never leaves the client side. The connection is made directly from client to client.

```
socket.on('key',function(data)
{
    console.log(data);
    console.log(yourName.val());
    console.log(hisName.val());
    if(myId == 1)
{
    console.log("roomKey" + roomKey);
        roomKey = data.key1;
```

```

}
else
{
    console.log("roomKey" + roomKey);
    roomKey = data.key2;
}
}
);

```

The script describes the client-side function that implements handshake. The generalized algorithm of the Cryp2Chat application is illustrated in Fig. 3.

As shown in the flowchart, from the moment of receiving the encrypted message and till the moment the message is sent to the recipient, the server undertakes the only action – certificate (i.e. license) verification. All the other steps associated with encryption, key generation, the transmission of the cryptogram to the recipient and decrypting of the cryptogram by the recipient, occur at the clients and in their browsers.

IV. EXPERIMENT PROCEDURE

Experimental study of the application was conducted on a typical mobile phone, where Cryp2Chat program was installed. Mobile phone is Nexus 5 with the processor speed 2260 MHz and with the operating system Android 4.4.4. This operating system supports novelties related to the safe operation in the browser. When a user opens an application, it verifies the certificate on the sender's device. In case of a successful verification the sender chooses a receiver. In case the connection is completed, the receiver's public key and a signature are taken from the browser local database, or they are requested from the server. Next the program encrypts the message and the sender's signature key. The message is sent to the server, and it verifies this signature on the basis of the contacts list. If the sender's signature exists in the server database, the latter immediately transmits the message to the recipient. In case of an incoming message the signature of the recipient is verified and it is decrypted with a secret decryption key.

The experimental results with Cryp2Chat prototype are shown on Figures 4 - 7.

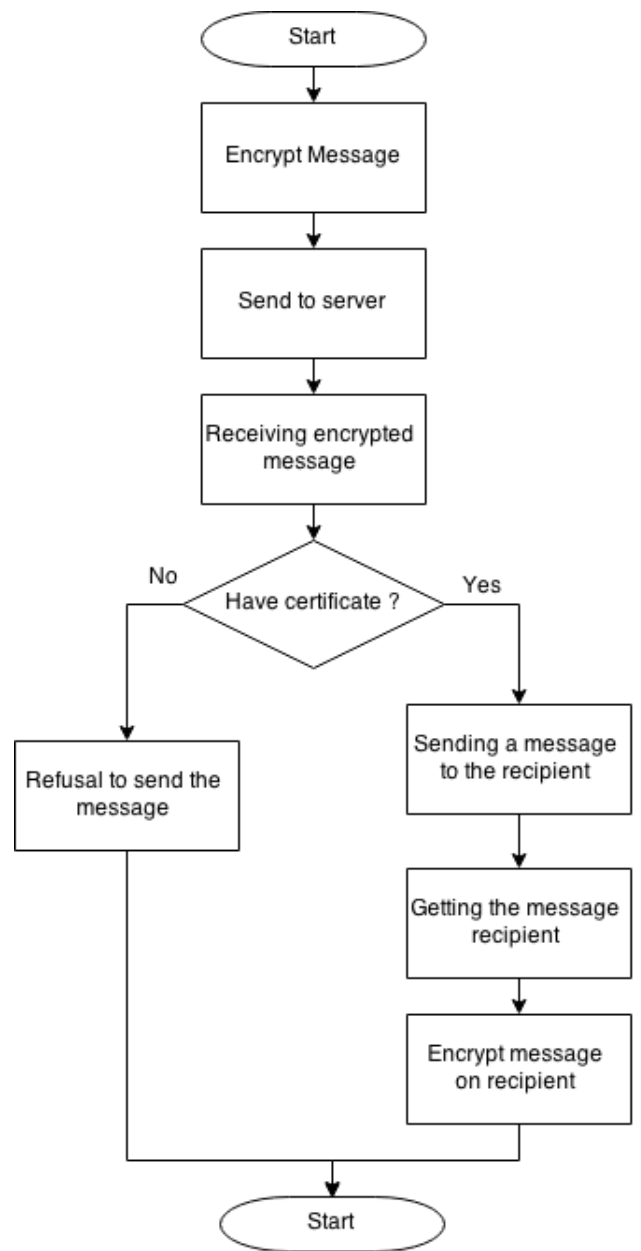


Fig. 3. Generalised Algorithm of Cryp2Chat Application

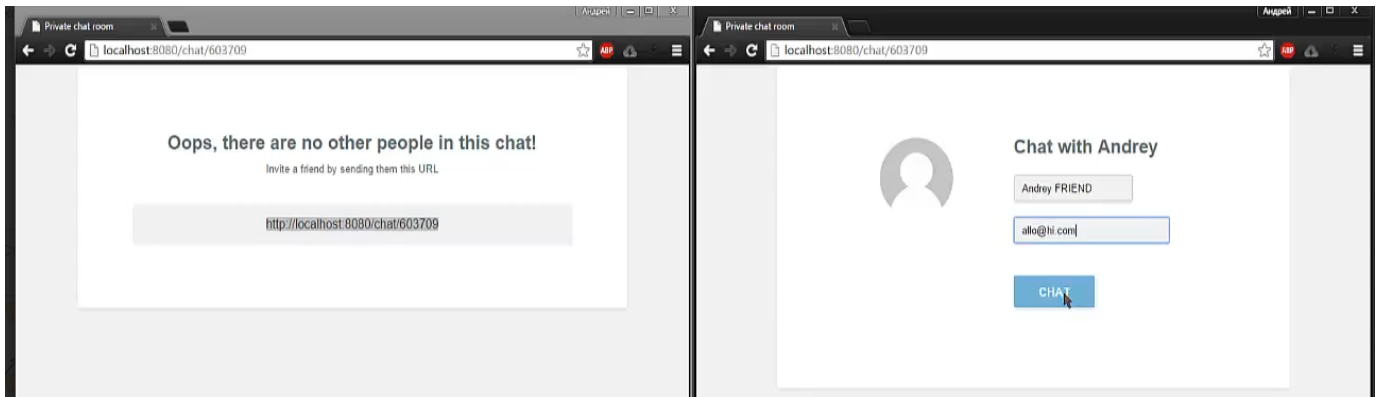


Fig. 4. Introducing the Users

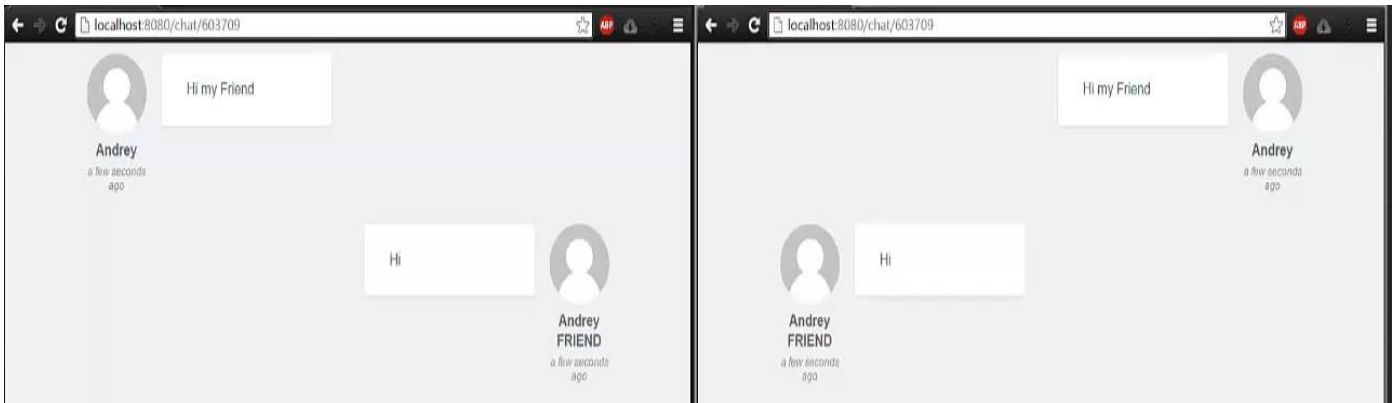


Fig. 5 Exchange with Test Messages



Fig. 6 Console of the First Client

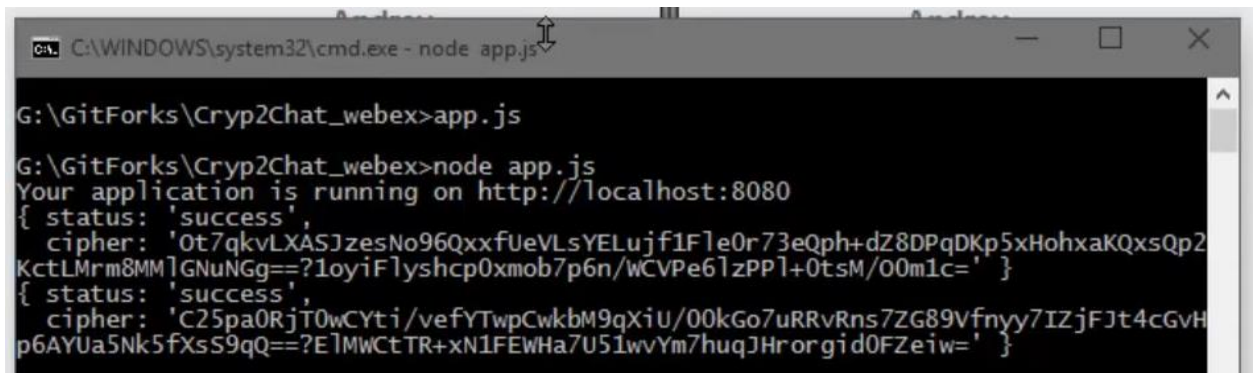


Fig. 7 Console of the Server

On the console of the first client and on the server console one could see only the encrypted string. This way the information is transmitted to the server (Fig. 7). Additionally, the recipient - the second client - is the only one who possesses the key to decrypt it.

Further we conducted an experiment for a group of 20 users. Specially for this purpose we launched the site in the cloud Azure that hosts Cryp2Chat application - <http://cryp2chat.azurewebsites.net/>. Based on the experiment we have had the following results:

- high speed of response from the client's side as well as from the server side;

- a sufficiently high contact capacity of the program, as all 20 users managed to establish contacts with their subscribers simultaneously.

Fig. 8 is a table of qualitative indicators of Cryp2Chat application along with its analogues. In the table the following conventional symbols are employed:

- v – activated functional features of the program,
- x – inactivated functional features of the program,
- * – business version. There exists a business version, but it is patented under a different name and it might be a slightly different product.





	 cryp2chat	 SnapChat	 WhatsApp	 Telegram
End2End encrypting	✓	✓	✓	✗
Support of crypto signs	✓	✗	✗	✓
Business version	✓	✓	*	✗
Oriented onto the Russian Federation and CIS market	✓	✗	✗	✓
Audio & Video translation	✓	✗	✓	✓
Crossplatform	✓	✗	✗	✓
Self-destruction of messages	✓	✗	✗	✓
The ability to work without a server	✓	✗	✗	✗

Fig. 8. Cryp2Chat Application and its Analogues

Figure 8 illustrates the following advantages of Cryp2Chat application:

- 1) The application corresponds to all the parameters;
- 2) It provides a cross-platform messaging and self-destruction of messages;
- 3) It uses translator servers, i.e. working on peer2peer scheme.

V. POTENTIAL DANGERS

While designing the application three possible potential dangers were considered:

1. Brute force. Kaspersky blog has been used to assess the possibility of selecting passwords [2]. The program has shown that the selection of the password with a key of about 50 characters length, including special characters, will take more than 100,000 years. Even on a powerful botnet Conficker a password will be sorted out for ten thousand centuries.

2. Key theft. It is impossible for two reasons:

- If it is *android* application, the "sandbox" - a tightly controlled set of resources for the execution of the guest program - will not give to another application access to the files with a password,

- If it is *web* application, the call to a variable is impossible, as a pointer to an element is deleted, and it is only the inner code that can refer to this variable.

3. The application code cannot be changed because:

- If it is web application, then the downloaded code is stored when you start the application for the first time and it cannot be downloaded when you run,

- If it is the native application, changes in a code from the server side does not lead to a change of the client application code.

The transfer of potentially dangerous information (acts of terrorism, drug sales) is prevented because control data exchange is carried out with the use of an electronic signature. While registering the user generates a signature. This is a RSA key that is passed to the server, stored there and never changed.

When sending a message, the server checks the signature and if this signature is missing on the server, this message is not sent. Also, the signature may be withdrawn from server storage due to violation of the license agreement or similar cases. Thus, it is possible also to control the transmission of messages. Though we do not know what is encrypted in the message, we may deny the user in the network communication services.

CONCLUSION

In the future, we plan to rewrite the project from scratch and to implement it as a complete business solution with further access to the market. Additionally we plan to develop graphical password and voice authentication function. In addition, the plan is to transfer video, audio and other files.

REFERENCES

- [1] Tor – The Onion Router. Wikipedia, the free encyclopedia/ URL: <https://ru.wikipedia.org/wiki/Tor#.D0.90>. [08.08.2014]
- [2] Blog.kaspersky URL: blog.kaspersky.com/password-check.
- [3] Tor: Overview URL: www.torproject.org/about/overview.html.en

Searching method of personal details on the basis of fuzzy comparison

Nataliia Limanova (*Author*)

Department of Technical Systems Software and
Management
Povolzhskiy State University of Telecommunications and
Informatics
Samara, Russia
Nataliya.I.Limanova@gmail.com

Maksim Sedov (*Author*)

Department of Technical Systems Software and
Management
Povolzhskiy State University of Telecommunications and
Informatics
Samara, Russia
SedovMN@inbox.ru

Abstract — During the information exchange from one department to another there is a problem of personal identification. This problem concerns the people who have partially or completely not coinciding personal details. In the represented work the new algorithm for identification of such people is elaborated. The algorithm is based on the fuzzy comparison and the metrics of Levenshtein. It allows us to find persons who have partial or complete not matching in surnames, names and other requisites in databases. The algorithm is implemented in PL-SQL in the Oracle database 11g.

Keywords — *Interdepartmental exchange of information, indistinct matching, search of personal details, function of intellectual matching, personal identification number (PIN).*

I. INTRODUCTION

In the course of the interdepartmental information exchange there is an approval problem of the main personal details (full name, birth date, address, passport data, etc.) in databases of various departments. The problem of personal identification has the greatest relevance for physical persons who have partially or completely not coinciding personal details.

For optimum control of big data files, in which the information about physical persons is included, it is necessary to provide centralized storage regulations of such personal details as full name, birth date, address, passport data, etc. Recently various departments – holders of local databases have aimed to combine these arrays for simplification and improvement of work quality. But there is a problem of personal details comparison in different databases. In such cases the elaborated intellectual algorithm of data search in databases or, in the other words, the algorithm of identification of physical persons comes to the aid.

For convenience of data processing, to each set of details the so-called personal identification number (PIN) is assigned. In the cases of handling or transferring of physical person data all binding is performed to this PIN. Unfortunately, in Russia, there is no uniform database with personal details of all residents, and therefore in each department the separate register of physical persons is kept, and own PINs are given.

The problem arises in the case of residents' information exchange between the organizations. So it is necessary to execute a binding of the entering personal data to the already available information. For an unambiguous binding it is necessary to execute intellectual search of physical person in base receiver which shall consider a set of factors: the mistakes in the case of manual input in the database, the absent or obsolete personal details and etc. It is reasonable to assume that similar search must be implemented in the form of the specialized software [1].

II. PROBLEM OF THE AUTOMATED SEARCH

Traditionally this problem is solved by the analysis of identity of the main personal details. There are several details: name, surname, middle name, date of birth, series, passport number and address. Having unambiguously determined coincidence of the existing and new details, it is possible to execute identification of personal details in a database [1][2]. This method of search is carried out manually only in that case when the amount of the transmitted data is small (number of personal details is no more than 30). In case of large volumes of transmitted data the computer comparison of identity of details is used. Such approach allows to determine on average (50 – 60) % of total number of identifiable personal details. The remained (40 – 50) % represent personal data in which the details in parts or in full don't match. It is more difficult to handle such information manually. Accordingly, the computer search task is divided into three subtasks depending on the type of input data. As a result of comparison the following three types of results can turn out.

1. The person is found. This conclusion can be created as a result of direct comparison of details, and equality of sets of certain key data. In this case the personal details becomes attached directly to the corresponding PIN.

2. The person is ambiguously determined. This result is displayed in the presence of mistakes, both in new data, and in the earlier received one. For example, the operator's mistakes in the case of manual input of the main details are possible, data corruption during transmission, incorrect work of package

requests in case of information processing, etc. In this case the list of PINs which main details are mostly approached to identifiable data is displayed.

3. The person isn't found. This case shows that this personal details is absent in the database and for a binding of this person to the PIN it is necessary to add him to the available data set with assignment of a new PIN.

When creating an automated complex software, which yields above-mentioned results, the most important was to determine borders between the first and second cases, and also between the second and third. The software working without similar differentiation, will put down PINs to all found persons unambiguously, and those whose data are determined ambiguously, are removed in the report for manual handling by the operator. Thus all not found persons will be added to base with assignment of a new PIN. Now let's imagine that in case of any discrepancy of the main details, the data will be provided to the report, or that is even worse, will be added as new. For example, the woman name is Natalia, she got married, respectively she has replaced her surname, she has moved to other residence and she has changed the passport. Besides, in the database she is registered under the name of Natalya, and in her birth date there is a mistake, an incorrectly specified number. When handling such data the program will decide that it is the new person and will add them with assignment of a new PIN. Of course, to a new PIN will set any task in compliance. As a result it turns out that data on one personal details are doubled and different PINs of one person operate with different tasks. If the error is not corrected immediately, then the number of incorrect data will grow up in the geometric progression. On correction of consequences of operation of such software a large number of competent employees of organization will spend a lot of time and forces [3][4][5].

The wrong identification can also lead to a large number of data in the report for manual working off, to assignment of the PIN to incorrect person and to addition of excessive data. At worst case the consequences of such mistakes can completely paralyze work of organization for indefinite time, at the best case – to take away more than 10% of working hours of specialists for errors correction. The analysis of the existing software showed that there is no single identifier; the universal algorithm of identification is also absent.

III. MATHEMATICAL MODEL OF SEARCH METHOD ON THE BASIS OF FUZZY COMPARISON

Some types of the metrics reflecting intuitive concept of similarity of lines are known. The most common are Hamming's distance, Levenshtein's metric and distance editing [6][7][8].

Hamming's distance is determined for lines of identical length and is set as number of line items in which symbols don't match. In fact, Hamming's distance is calculated as minimum price of transformation of one line in another when

the only one transaction of editing lines – replacement is possible.

In a case when it is required to make comparison of lines of different length, Levenstein's metrics or distance editing are used. These two metrics are very similar on creation and actually are the same metrics, little modified for each case. For example, Levenstein's metrics is determined as minimum price of transformation of one line in another with the use of three transactions: inserts, replacements and removals of a symbol, and all three transactions have identical weight.

The distance editing is modification of Levenstein's metrics in the case when only two transactions are allowed: insert and removal.

Due to the above, Levenstein's general metrics which supports all three transactions with line was chosen. For further operation the linguistic variable "similarity of lines" was constructed. It is decided to allocate the following terms: "lines match", "lines almost match", "lines are similar", "lines are similar and dissimilar at the same time", "lines aren't similar".

In the result of the analysis of functions of accessory of linguistic terms there was a need to modify the method of calculation of Levenstein's metrics. It was required to modify metrics so that the distance between lines depended on length of the compared lines.

Theorem 1:

We will designate by means of size $p(s_1, s_2)$ Levenstein's metrics, and size $\|s_i\|$ – length of line s_i . Then function:

$$r(s_1, s_2) = \frac{p(s_1, s_2)}{\max\{\|s_1\|, \|s_2\|\}}, \quad (1)$$

is a metrics.

Proof (not strict proof):

Because $p(s_1, s_2)$ is a metrics, we have:

$$\begin{aligned} p(s_1, s_2) &\geq 0, \\ p(s_1, s_2) &= p(s_2, s_1), \\ p(s_1, s_2) + p(s_2, s_3) &\geq p(s_1, s_3) \end{aligned}$$

for any lines s_1, s_2 and s_3 . Considering these ratios and equality (1), we come to a conclusion that $r(s_1, s_2)$ satisfies to the first two axioms determining metrics. It is necessary to prove that for any lines s_1, s_2 and s_3 function $r(s_1, s_2)$ satisfies to a triangle inequality:

$$r(s_1, s_2) + r(s_2, s_3) \geq r(s_1, s_3).$$

Write this inequality in the form:

$$\frac{p(s_1, s_2)}{\max\{\|s_1\|, \|s_2\|\}} + \frac{p(s_2, s_3)}{\max\{\|s_2\|, \|s_3\|\}} - \frac{p(s_1, s_3)}{\max\{\|s_1\|, \|s_3\|\}} \geq 0.$$

The following cases are possible:

1. $\|s_1\| \leq \|s_2\| \leq \|s_3\|$
2. $\|s_2\| \leq \|s_3\| \leq \|s_1\|$
3. $\|s_3\| \leq \|s_1\| \leq \|s_2\|$
4. $\|s_2\| \leq \|s_1\| \leq \|s_3\|$

5. $\|s_1\| \leq \|s_3\| \leq \|s_2\|$
6. $\|s_3\| \leq \|s_2\| \leq \|s_1\|$

Consider the first case. We have:

$$\frac{p(s_1, s_2)}{\max\{\|s_1\|, \|s_2\|\}} + \frac{p(s_2, s_3)}{\max\{\|s_2\|, \|s_3\|\}} - \frac{p(s_1, s_3)}{\max\{\|s_1\|, \|s_3\|\}} = \frac{p(s_1, s_2)}{\|s_2\|} + \frac{p(s_2, s_3)}{\|s_3\|} - \frac{p(s_1, s_3)}{\|s_3\|} \geq 0.$$

Thus, for the first case the triangle inequality is carried out. As the second case is similar to the first one, based on similar calculations we draw a conclusion that for the second case the triangle inequality is also carried out.

We will turn to consideration of the third case. So, in the third case we have:

$$\begin{aligned} r(s_1, s_2) + r(s_2, s_3) - r(s_1, s_3) &= \\ &= \frac{1}{\|s_2\|} (r(s_1, s_2) + r(s_2, s_3)) - \frac{1}{\|s_1\|} r(s_1, s_3). \end{aligned} \quad (2)$$

We'll consider a question when the minimum of the function which is in the right part of this equality is reached. It is clear that if expression of $r(s_1, s_2) + r(s_2, s_3)$ reaches the minimum, and $r(s_1, s_3)$ reaches the maximum, the value of all expression will be minimum. The two specified conditions can be satisfied at the same time if two following statements are carried out at the same time:

- lines s_1 and s_3 have no common symbols,
- lines s_1 and s_3 are included as sublines in s_2 . Then:

$$\begin{aligned} r(s_1, s_3) &= \max\{\|s_1\|, \|s_3\|\} = \|s_1\|, \\ r(s_1, s_2) &= \|s_3\| + \|C\|, \quad r(s_2, s_3) = \|s_1\| + \|C\|, \end{aligned}$$

thus, the minimum value of expression (2) will register in a form:

$$\frac{\|s_3\| + \|C\| + \|s_1\| + \|C\|}{\|s_3\| + \|s_1\| + \|C\|} - \frac{\|s_1\|}{\|s_3\| + \|s_1\| + \|C\|} = \frac{\|C\|}{\|s_3\| + \|s_1\| + \|C\|} \geq 0.$$

Therefore, in the third case for function $r(s_1, s_3)$ a triangle inequality is also carried out. Other cases are similar to the already considered. Thus, function $r(s_1, s_2)$ is the metrics, defined in the set of lines. The theorem is proved.

Note: function $r(s_1, s_2)$ belongs to the interval $[0, 1]$ for any lines s_1 and s_2 .

In the offered algorithm this metrics is applied for operation with line personal details which includes full name, address, document, etc. Therefore the linguistic variable constructed with use of this metrics allows to process requests of search for the person similar to other person in details. Having accepted such request from the user, we actually receive two values: the value of a required detail and the radius of search.

IV. ALGORITHM OF SEARCH METHOD ON THE BASIS OF FUZZY COMPARISON

The Fig. 1 shows the integrated flowchart of developed algorithm of search method on the basis of fuzzy comparison. The offered algorithm presented in the form of process of Data Mining includes the following stages [9]:

1. analysis of subject domain;
2. problem definition;
3. preparation of data;
4. creation of models;
5. check and assessment of models;
6. model choice;
7. application of model;
9. correction and updating of model.

Consider these steps in details.

1. The subject domain represents data sets with the main personal details in the different organizations and departments.

2. The task of search consists in that in the conditions of absence of single personal identification number to search of a set of details in one database according to personal details in other database.

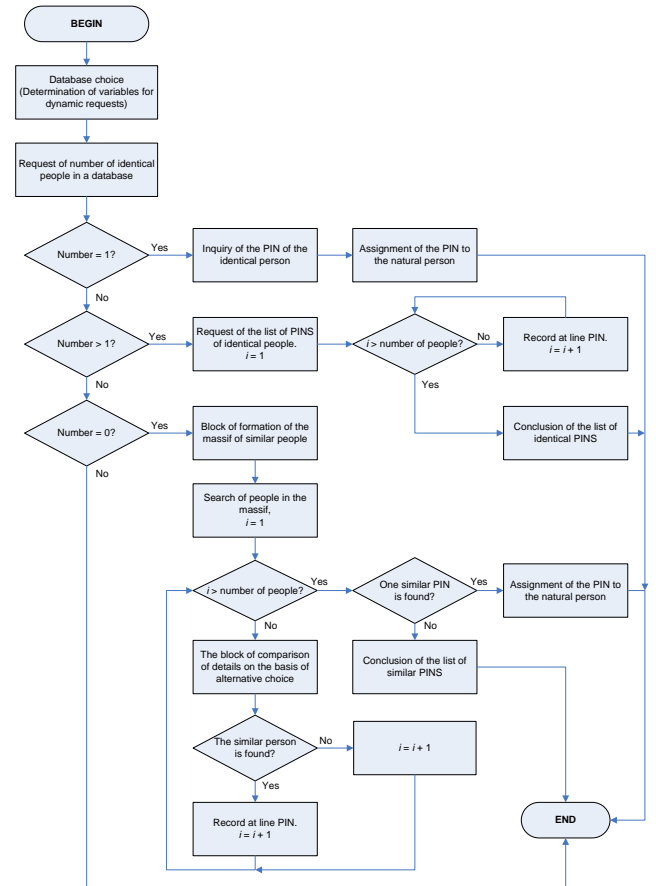


Fig. 1. The integrated flowchart of developed algorithm of search method on the basis of fuzzy comparison.

3. Preparation of data represents the organization of the integrated selection including about 300-500 sets, remotely similar to the required. The code fragment allowing to organize programmatically such selection is given below:

```
CURSOR persons
IS
SELECT p.person_id, p.lastname, p.firstname,
       p.patronymic, p.birthdate
FROM work.person p WHERE
(((SOUNDEX(TO_TRANSLIT(p.lastname)) =
SOUNDEX(TO_TRANSLIT(fo_Lastname)))
AND (SOUNDEX(TO_TRANSLIT(p.firstname)) =
SOUNDEX(TO_TRANSLIT(fo_Firstname))))
OR ((SOUNDEX(TO_TRANSLIT(p.lastname)) =
SOUNDEX(TO_TRANSLIT(fo_Lastname)))
AND (SOUNDEX(TO_TRANSLIT(p.patronymic)) =
SOUNDEX(TO_TRANSLIT(fo_Patronymic))))
OR ((SOUNDEX(TO_TRANSLIT(p.firstname)) =
SOUNDEX(TO_TRANSLIT(fo_Firstname)))
AND (SOUNDEX(TO_TRANSLIT(p.patronymic)) =
SOUNDEX(TO_TRANSLIT(fo_Patronymic)))));
```

4. Creation of models consists in detection of regularities in the analysis of the data obtained in result of step 3 shown in this data set and, perhaps suitable for future sets.

5. Check and assessment of models represent testing of regularities for quantity of the data sets satisfying with them. The more sets are suitable for specific model, the more valuable is revealed regularity.

6. The choice of model consists in detection of the most significant regularities for further use in case of future starts of identification procedure.

7. Application of model represents the use of the regularity received and approved in case of last start of identification procedure in the current data sets.

8. Correction and updating of model consist in the analysis of result of regularity appendix to a new data set, and, if necessary, correction of model for circle expansion of suitable sets by fuzzy search of compliance of personal details.

Programmatically it looks approximately like this (with use of dynamic SQL):

```
-- Perform fast identification
OPEN cur_Ref_fast_ident
FOR 'SELECT t.||v_Col_pin||
FROM '||v_Table||' t
WHERE UPPER(TRIM(t.||v_Col_lastname||)) =
UPPER(TRIM('||fo_Lastname||'))
AND UPPER(TRIM(t.||v_Col_firstname||)) =
UPPER(TRIM('||fo_Firstname||'))
AND NVL(UPPER(TRIM(t.||v_Col_patronymic||')), '-')
= NVL(UPPER(TRIM('||fo_Patronymic||')), '-')
AND t.||v_Col_birthdate||' =
'||TO_CHAR(fo_Birthdate, 'dd.mm.yyyy')||''';
FETCH cur_Ref_fast_ident BULK COLLECT
```

```
INTO c_fast_ident;
CLOSE cur_Ref_fast_ident;

-- Depending on the number of pins of identical persons
IF (NVL(c_fast_ident.count, 0) = 1) THEN
  fout_Pin := c_fast_ident(1);
ELSEIF (NVL(c_fast_ident.count, 0) > 1) THEN
  FOR i IN c_fast_ident.first..c_fast_ident.last LOOP
    fout_Pin_list:=fout_Pin_list||TO_CHAR(c_fast_ident(i))||' ';
  END LOOP;

-- If fast identification didn't yield results
ELSIF (NVL(c_fast_ident.count, 0) = 0) THEN

  -- write down data from the cursor in collection
  OPEN cur_Ref_full_ident FOR v_Cur_ident;
  FETCH cur_Ref_full_ident BULK COLLECT
  INTO c_full_ident;
  CLOSE cur_Ref_full_ident;
  IF (NVL(c_full_ident.count, 0) > 0) THEN
    FOR i IN c_full_ident.first..c_full_ident.last LOOP

      -- Perform complete identification
      -- The block of comparison of details on the basis of
      alternative choice (see Fig. 1)
      CASE
      ...
      WHEN (UPPER(TRIM(c_full_ident(i).ima)) =
UPPER(TRIM(fo_Firstname))
AND UPPER(TRIM(c_full_ident(i).oth)) =
UPPER(TRIM(fo_Patronymic))
AND ((analyzer_two_number(TO_NUMBER
(TO_CHAR(c_full_ident(i).dtr, 'ddmmyyyy')),
TO_NUMBER(TO_CHAR(fo_Birthdate, 'ddmmyyyy')))) = 1
AND analyzer_two_number(c_full_ident(i).nom,
fo_Passport_number) = 1) OR
((analyzer_two_number(TO_NUMBER
(TO_CHAR(c_full_ident(i).dtr, 'ddmmyyyy')),
TO_NUMBER(TO_CHAR(fo_Birthdate, 'ddmmyyyy')))) = 1
OR analyzer_two_number(c_full_ident(i).nom,
fo_Passport_number) = 1)
AND c_full_ident(i).dom = fo_House
AND c_full_ident(i).kva = fo_Flat)))
      THEN fout_Pin_list :=
fout_Pin_list||TO_CHAR(c_full_ident(i).pin)||' ';
      ...
      WHEN (UPPER(TRIM(c_full_ident(i).fam)) =
UPPER(TRIM(fo_Lastname))
AND UPPER(TRIM(c_full_ident(i).ima)) =
UPPER(TRIM(fo_Firstname))
AND analyzer_two_string(c_full_ident(i).oth,
fo_Patronymic) = 1)
      THEN v_Pin_list_sim :=
v_Pin_list_sim||TO_CHAR(c_full_ident(i).pin)||' ';
      ...
      ELSE NULL;
      END CASE;
    END LOOP;
  END IF;
END IF;
```

In developed implementation of algorithm in PL-SQL DBMS Oracle 11g [10] language, key functions are allocated for logically selected procedures ANALYZER TWO STRING and ANALYZER TWO NUMBER, created on the basis of the modified method calculation of Levenstein's metrics which allow carrying out intellectual comparison of two similar lines or numbers, taking into account possible inaccuracies or errors of input. These procedures can be applied not only for identification of details, but also everywhere where full text search with fuzzy set input data is required.

V. TECHNICAL AND ECONOMIC INDICATORS OF PROPOSED ALGORITHM

For the comparative analysis of developed algorithm let's consider technology of identification on the basis of direct comparison. When using this technology the emphasis goes on speed of records handling, but not on quality of decision making by system. As a result, after completion of procedure on the basis of direct comparison, there are many data (about 20-30% of total quantity of the lines) not connected with initial which need to be fulfilled manually that is extremely difficult in the case of large volumes of the processed data.

When comparing working indicators of two algorithms it is revealed:

Algorithm of direct comparison:

Data processing speed: ~ 100 000 lines per hour;

Identification accuracy (probability of exact searching method): ~ 80%

Algorithm of identification on the basis of fuzzy comparison:

Data processing speed: ~ 80 000 lines per hour;

Identification accuracy (probability of exact searching method): ~ 99,9%

It is possible to draw a conclusion that, operator's work in manual operation of results is minimized in developed algorithm i.e. though the speed of handling is slightly less, but the algorithm allows to significantly unload operators at the expense of intellectual system of decision making that can't offer algorithm of direct comparison.

When comparing economic characteristics of the developed software on the basis of described algorithm with procedure of direct comparison for annual amount of identification of 1 200 000 physical persons the following data were obtained: labor costs on information processing by the method of fuzzy comparison in comparison with method of direct comparison are reduced by 6,7 times, absolute decrease in labor costs constituted 1 446 hours, annual costs when using the fuzzy comparison method decreased by 3 times in comparison with the similar period of application of the direct comparison method, annual economic effect exceeded 580 000 rub. For descriptive reasons some cost indicators which are created when using the software developed and applied are displayed on the chart provided on Fig. 2. Sizes of costs are postponed on ordinate axis in rubles.

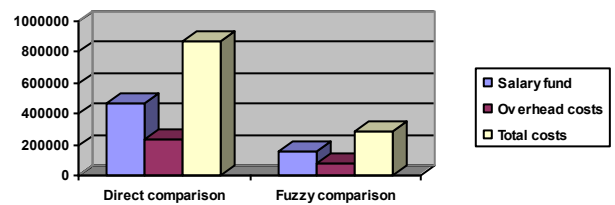


Fig. 2. The chart for the comparative analysis of cost indicators when using methods of direct and fuzzy comparison.

VI. CONCLUSIONS

The considered method of the computer search of personal data on the basis of fuzzy comparison designed with use of the Data Mining technology allows to quickly determining people, using data of search carried out earlier. The built-in system of details priority allows to identify person in such cases as change of surname, name, moving, mistakes in case of manual data entry, and in case of partially absent details.

Self-training systems allow releasing human resources for accomplishment of creative tasks. In this area the Data Mining technology provides a full range of theoretical and practical means for choice, development or use of intellectual computer systems.

The procedure of identification from this article can be considered as part of the system of decision support. The procedure does not require the operator intervention, gains experience and learns in the process of operation, allowing to completely exempt specialists from low-profile, inefficient, manual operation directly with the sets of personal details which are stored in databases.

The developed algorithm shows good results when fields with different information inside (name, address, postcode, phone etc) are compared. Indeed, any symbolical value, whether it be full name, number of the passport or address, it is possible to present in the form of string. In the course of two strings comparison with the help of the offered algorithm, the distinctions of these lines are revealed, such as the admissions of separate symbols or incorrect single symbols which can arise at typographical errors in a manual data set. I.e., from the point of view of symbol-to-symbol comparison, there is no difference between comparison of two passport numbers or two surnames.

In long terms, this algorithm has the possibility of successful implementation in systems of global merger of storages of the state or commercial organizations, for maintaining a single database of the population of any country of the world. The logical structure of developed algorithm allows realizing it in any popular programming language. Features of algorithm allows applying program procedures on its basis both in small organizations, and in large corporations,

everywhere, where the register of physical persons data is conducted and staticized. Possible examples of use: portal of state services, medical electronic systems, personnel and accounting systems of accounting of employees, bank systems of data storage on clients, etc.

The algorithm is realized in PL-SQL of Oracle 11g database management system. The developed software realizing a method of the computer search of personal data on the basis of fuzzy comparison is implemented and successfully operates since 2007 in the municipal institution «City information center» Togliatti city of Samara region.

REFERENCES

- [1] Selection of materials on the international experience of legislative regulation of use of systems of the personality's identification (<http://www.kongord.ru/Index/Prison/SViP.htm>).
- [2] The report on accomplishment of research, developmental work "Development of mechanisms of unambiguous identification of data on the physical persons and real estates which are stored in various information systems of public authorities and local government (http://www.nisse.ru/business/article/article_464.html).
- [3] Regulations on personal identification number of the citizens of the Russian Federation living or staying in the territory of St. Petersburg (<http://iac.spb.ru/shablon.asp?subpage=171&id=40&dir=0>).
- [4] The "Moscow Social Card" project (<http://www.soccard.ru>).
- [5] The collection of theses of city scientific and practical conference of students, graduate students, teachers of higher education institutions and specialists of local government offices of Tolyatti "Informatization in the social sphere" (<http://it-exclusive.ru/idperson/docs/stat.doc>).
- [6] Hamming R. V. The theory of coding and the theory of information, trans. Edited by BS Tsybakov, Radio and Communications, 1983.
- [7] Levenstein V. I. Binary codes with correction of losses, inserts and replacements of symbols, reports of Academy of Sciences of the USSR vol.163, 1965.
- [8] Boytsov L.M. Analysis of lines, http://itman.narod.ru/articles/infoscope/string_search.1-3.html.
- [9] Chubukova I.A., "Data Mining", training course, publishing house of Internet university of information technologies (<http://www.intuit.ru/>), 2006.
- [10] Scott Urman, "ORACLE 9i - Programming in PL / SQL", tutorial, Oracle Press – publishing house "Lory", 2004.

Seamless Development Applicability: an Experiment

Alexandr Naumchev
Software Engineering Laboratory
Innopolis University
Kazan, Russia
Email: a.naumchev@innopolis.ru

Abstract—Requirements and code, in conventional software engineering wisdom, belong to entirely different worlds. Is it possible to unify these two worlds? A unified framework could help make software easier to change and reuse. To explore the feasibility of such an approach, the case study reported here takes a classic example from the requirements engineering literature and describes it using a programming language framework to express both domain and machine properties. The paper describes the solution, discusses its benefits and limitations, and assesses its scalability.

Keywords—software engineering, requirements specifications, multirequirements, Eiffel

I. INTRODUCTION

Nowadays the dominating view on the software engineering discipline includes an implicit assumption that engineering the requirements, designing the architecture and implementing the code are all separate activities. “Separate” means that an engineer performs only one of them at the same time and produces different artifacts as the output. This implicit assumption is cultivated by the top software engineering schools who promote the idea explicitly enough to push it to the students’ subconscious level.

A. Problems with the Current Approach

The usual view in software engineering considers requirements documents and source code as different artifacts, under the responsibility of different people. This approach, however, introduces communication overhead, and raises the question of how to keep the various artifacts consistent when either of them needs to change. A change introduced to any of the mentioned artifacts needs to be synchronized with the others. At some point the control is inevitably lost: for example, a critical bug is found during the software operation, and the software developers dig into the fixing process directly, because there is no time to wait until the requirements analysts and system architects update their documents to let the developers actually fix the problem. The problem is partially solved with complicated configuration management, which is expensive and difficult to maintain, and may serve as a source of evil as well: there are so called “technical commits”. Only senior developers are allowed to make them, and the basic idea is that such commits do not have to be linked to some task, bug or user story (if the team practices Agile). Quite often the technical commits contain basically whole new features or big chunks of code not linked to any document.

Why should we try to minimize gaps between requirements and code? At the very least because successful software

evolves. The customers want more features, they want to improve existing features, and they want to know how much money it will cost and how much time it will take. If it is possible to relate the ideas to the artifacts, then by comparing complexity of some new idea with an existing one, already implemented, it will be possible to estimate the resources required for implementing the new idea.

The list of the problems discussed above does not pretend to be exhaustive of course, but it should be sufficient to start thinking about changing the overall approach.

B. Existing Solutions

Typically the problems from Section I-A are resolved by carefully choosing appropriate notations for every development life cycle phase. The selection criteria include possibility of establishing traceability links between different notations. Each phase requires the output of the previous phase on its input and on its output produces the input for the next phase. An example of applying this approach is given in work [2]. This work also contains an overview of the most popular notations used in formal software development. For instance, the software development case described in work [2] uses natural language for requirements document, RSML [7] for specification document, Event-B [1] for developing software formal model, formalizing the requirements and formally verifying the model against the requirements. Finally EB2C tool [8] generates executable source code taking the formal model expressed in Event-B. For moving from the requirements document to the specification document the Problem Frames Approach [5] is applied. The latter method produces a problem frames model on the output.

Needles to say, such approach requires people with very rich set of skills: for example, to produce a specification document expressed in RSML, the responsible person also has to understand the Problem Frames Approach. In a similar fashion the person responsible for modeling in Event-B also has to be proficient with RSML, and so on.

As a software engineer we should not forget why there is a huge gap between requirements and code at all. The fundamental reason is in limited expressive power of programming languages compared to expressive power of any natural language. That is why there are many “intermediate” notations serving for smooth transition from natural language requirements to source code; that is why the coding phase and the requirements engineering phase typically have tiny overlaps in time, and there are other software development life cycle phases between them. If it was possible to express any executable requirement using a subset of some programming language, than the problem would disappear.

C. Unified View on Software: The Hypothesis

It is possible to design such a software development process that:

- 1) By specifying the requirements the analyst at the same time will also design the solution
- 2) The resulting document may be linked in an intuitive way to an algorithmic implementation
- 3) The resulting implementation will be formally provable against the requirements specification
- 4) Small change in the requirements specification will cause proportionally small change in the design and the implementation

Parts 1, 2 and 3 promote consistency between the requirements, design and implementation; part 4 promotes predictability of resources estimations.

D. How to Test the Hypothesis

The following process seems to be feasible for testing adequacy of the stated hypothesis:

- 1) Propose a candidate process
- 2) Select some real projects which are presumably prone to the problems stated in section I-A
- 3) Apply the proposed process to the selected projects and see how it goes

In [11] Meyer sketched such a process based on using object orientation for representing the relationships between the conceptual objects mentioned in the requirements document. The basic idea was to have an object-oriented code along with the natural language description of a requirements item. Each code fragment in its turn may be represented graphically as a BON diagram [15].

The main problem with [11] was the example used for the demonstration purposes: it was self-referential. That is, it contains "requirements for the requirements".

Nevertheless, the work [11] demonstrates that object orientation contributes to understanding the relationships between the objects. However, requirements (in their general form) are beyond this: to specify requirements, as described by Jackson and Zave in [6], is also to specify all allowed sequences of events associated with a given problem area.

This work provides an example of how one could combine approaches from [11] and [6] by adding fully-fledged contracts, both in their classical and model-based semantics, to the requirements specification notation. More precisely, it contains every requirements item from the Zoo Turnstile example discussed in [6] represented using the model-based [13] contracts-equipped [10] object-oriented [9] notation (Eiffel).

II. THEORETICAL AND TECHNICAL BACKGROUND

A. Design By Contract

A comprehensive description of Design By Contract is given in [10]. Design By Contract integrates Hoare-style assertions [3] within object-oriented programs [9]. This concept assumes that each class feature (member), is equipped with its pre- and postcondition. The postcondition has to hold

whenever the precondition held and the feature finished its computation before the next feature is invoked. The class itself is equipped with an invariant expression which holds in all states of the corresponding instantiated objects.

B. Model-Based Contracts

If classical contracts are for constraining the data actually held by run-time objects, model-based contracts are "meta" contracts for constraining the objects as mathematical entities (sets, sequences, bags, relations etc.), and the corresponding mathematical representations are not actually instantiated at run-time as parts of the objects. Model-Based Contracts are needed when it is not possible to capture all the nuances by means of classical contracts. Some examples of such situations and a comprehensive description of the concept is given in the PhD thesis [13].

C. AutoProof

Object-oriented classes constrained with contracts (both classical and model-based) may be formally verified using an automation called AutoProof [14]. AutoProof traverses over the class features and proves formally that the precondition conjuncted with the class invariant ensures the postcondition together with the class invariant after the feature application. If all the class features are verified, then the class is considered verified.

III. UNIFYING THE TWO WORLDS: AN EXAMPLE

This section shows the approach at work. It takes the example introduced by Jackson and Zave in [6] in 1995. Originally this example was used to demonstrate the process of deriving specifications from requirements, and the unified approach captures all the nuances of this process.

A. Example Overview

The authors of [6] start with giving the overall context: "...Our small example concerns the control of a turnstile at the entry to a zoo. The turnstile consists of a rotating barrier and a coin slot, and is fitted with an electrical interface..." This small paragraph describes mostly relationships between the conceptual objects and thus may be expressed in the style of work [11]:

```
class ZOO
feature
  turnstile: TURNSTILE
end

class TURNSTILE
feature
  coinslot: COINSLOT
  barrier: BARRIER
invariant
  coinslot.turnstile = Current
  barrier.turnstile = Current
end

class COINSLOT
feature
```

```

    turnstile: TURNSTILE
invariant
    turnstile.coinslot = Current
end

class BARRIER
feature
    turnstile: TURNSTILE
invariant
    turnstile.barrier = Current
end

```

Translating this code back to English using the object-oriented semantics results in almost the same initial description: "A ZOO has a TURNSTILE turnstile; a TURNSTILE has a COINSLOT coinslot and a BARRIER barrier so that coinslot has Current TURNSTILE as turnstile and barrier has Current TURNSTILE as turnstile..." COINSLOT and BARRIER hold references to the TURNSTILE instances in order to capture the "electrical interface" phenomena: the word "interface" means something over which the parties are able to communicate with each other; communicating means sending messages to each other, and to send message to someone in the object-oriented world is to take the corresponding instance and perform a qualified call. So at the very least the parties should hold references to each other to be able to communicate in two directions.

B. The Designation Set

After stating the problem context the authors of [6] describe a *designation set*. Each designation basically corresponds to a separate type of events observed in the problem area. The designations are provided in form of the predicates:

- **Push**(*e*): In event *e* a visitor pushes the **barrier** to its intermediate position
- **Enter**(*e*): In event *e* a visitor pushes the barrier fully home and so gains entry to the **zoo**
- **Coin**(*e*): In event *e* a valid coin is inserted into the **coin slot**
- **Lock**(*e*): In event *e* the **turnstile** receives a locking signal
- **Unlock**(*e*): In event *e* the **turnstile** receives an un-locking signal

The representation of this designation set provided below uses Eiffel features names as labels for the events types (entities introduced earlier are not repeated afterwards). The aforementioned natural language descriptions provide heuristics on which feature should be added to which class (the association is highlighted with **bold**). Not only different types of events, but also the history of the corresponding events, are designed using Eiffel features. For example, *enters* : *MML_SEQUENCE*[*INTEGER_64*] is a sequence of moments in time expressed in milliseconds when events of type *enter* took place. *model* annotation says that *enters* feature will be used for expressing the model-based part of the contract (model-based contracts were introduced in section II-B). *MML_SEQUENCE* is a class from the *MML* (Mathematical

Modeling Library) and denotes mathematical sequence. *MML* was designed specially to express model-based contracts. Although it is possible to instantiate some simple objects from these classes (like a sequence containing one element), one cannot modify the instances.

```

note
    model: enters
deferred class ZOO
feature
    enter
deferred
ensure
    enters.but_last ~ old enters
    enters.last > old enters.last
end
    enters: MML_SEQUENCE[INTEGER_64]
end

note
    model: locks, unlocks
deferred class TURNSTILE
feature
    lock
deferred
ensure
    locks.but_last ~ old locks
    locks.last > old locks.last
end
    unlock
deferred
ensure
    unlocks.but_last ~ old unlocks
    unlocks.last > old unlocks.last
end
    locks: MML_SEQUENCE[INTEGER_64]
    unlocks: MML_SEQUENCE[INTEGER_64]
end

note
    model: coins
deferred class COINSLOT
feature
    coin
deferred
ensure
    coins.but_last ~ old coins
    coins.last > old coins.last
end
    coins: MML_SEQUENCE[INTEGER_64]
end

note
    model: pushes
deferred class BARRIER
feature
    push
deferred
ensure
    pushes.but_last ~ old pushes
    pushes.last > old pushes.last
end

```

```

pushes : MML_SEQUENCE[INTEGER_64]
end

```

The *deferred* keyword is used to highlight that the events are only specified formally, without specifying the corresponding operational reactions of the software to the events. The *ensure* clause is used to specify what conditions should be satisfied after reacting on an event. These specifications are intuitively plausible: the events history should be complemented with the new event occurrence, and the time of the new event should be strictly bigger than the time of the previous event.

C. Shared Phenomena

The authors of [6] introduce the notion of shared phenomena - that is, the phenomena visible to both the world and the machine (the notions of the world and the machine were introduced by Jackson in [4]). In the present approach this notion is covered by using the "has a" relationships between the *ZOO* and the *TURNSTILE* classes, accompanied with the model-based contracts. Namely, since a *ZOO* has a turnstile as its feature, it can see any phenomena hosted by the turnstile: *locks*, *unlocks*, *coins*, *pushes*. And since a *TURNSTILE* does not hold any references to a *ZOO*, it can not observe nor control the *enter* events modeled by *ZOO*.

D. Specifying the System

All the properties of the problem derived in [6] - be they optative or indicative descriptions - can be conceptually divided into the two main categories.

1) *Properties which hold at any moment in time*: An example of such properties is the *OPT1* requirement saying that entries should never exceed payments (the authors of [6] use *OPT** for labeling properties expressed in an optative mood). Within the present approach this requirement can be expressed in the following way:

```

deferred class ZOO
feature
  enters : MML_SEQUENCE[INTEGER_64]
  turnstile : TURNSTILE
invariant
  enters.count <= turnstile.
    coinslot.coins.count
end

```

The "something always holds" semantics fits perfectly into the semantics of Eiffel invariant: "something holds in all states of the object".

2) *Properties which hold depending on the type of the next event to occur*: The indicative property *IND2* saying that it is impossible to push the barrier if the turnstile is locked will serve as an example. Below is the corresponding specification:

```

deferred class BARRIER
feature
  push
require
  not turnstile.unlocks.is_empty
  (not turnstile.locks.is_empty)
  implies

```

```

    turnstile.unlocks.last >
      turnstile.locks.last
deferred
end
pushes : MML_SEQUENCE[INTEGER_64]
end

```

The initial description is divided into the two different claims: first, the turnstile should be unlocked at least once, and second, if the turnstile has ever been locked, the last unlock should have occurred later than the last lock.

3) *Real Time Properties*: The authors of [6] derive several timing constraints on the events. For example, the *OPT7* requirement says that the amount of time between the moment when the number of the barrier pushes becomes equal to the number of coins inserted and the moment when the turnstile is locked should be less than 760 milliseconds. It is possible to make this property finer grained. First, if after the next *push* event the number of pushes becomes equal to the number of coins, then after reacting on the *push* event the turnstile should be locked at some point before the next *push* event occurs:

```

class BARRIER
feature
  turnstile : TURNSTILE
  push
deferred
ensure
  ((old turnstile.unlocks.last >
    old turnstile.locks.last) and
    (pushes.count =
      turnstile.coinslot.coins.count))
    implies turnstile.locks.last >
      pushes.last
end
pushes : MML_SEQUENCE[INTEGER_64]
end

```

Second, if the last *lock* event occurred later than the last *push* event, then the time distance between them is smaller than 760:

```

class TURNSTILE
feature
  barrier : BARRIER
  locks : MML_SEQUENCE[INTEGER_64]
  unlocks : MML_SEQUENCE[INTEGER_64]
invariant
  locks.last > barrier.pushes.last
  implies
    (locks.last - barrier.
      pushes.last) < 760
end

```

E. Specifying the "Unspecifiable"

One of the requirements mentioned in [6] was *OPT2* saying that the visitors who pay are not prevented from entering the Zoo. The authors give only informal statement of this requirement:

$$\forall v, m, n \bullet ((Enter\#(v, m) \wedge Coin\#(v, n) \wedge (m < n)) \Rightarrow$$

'The machine will not prevent another Enter event'

The antecedent of this implication should be read like "number of entries is less than the number of coins inserted". In the present specification system this requirement can be formalized easily:

```
deferred class ZOO
feature
  enter
  require
    enters.count <
      turnstile.coinslot.coins.count
  deferred
  end
  enters: MML_SEQUENCE[INTEGER_64]
end
```

It works because semantically the *require* clause specified above is the strongest precondition of the *enter* feature. That is, if some class inherits from *ZOO* and redefines the *enter* feature, it will be allowed to redefine the precondition by using only the *require else* clause that weakens the precondition by "or"-ing it with the original one. And so, if the *enters.count < turnstile.coinslot.coins.count* condition is satisfied, the precondition of the *enter* feature will always be satisfied, thus allowing an *enter* event to occur.

Not only this specification formalizes *OPT2* - it also ensures satisfaction of *OPT1* (together with the *ensure* clause for the *enter* feature introduced earlier): indeed, if the number of enters is always strictly smaller than the number of coins inserted before any *enter* event occurrence, then after the *event* occurrence the number of entries will not exceed the number of coins inserted.

In the process of research the author of the present work identified that the aforementioned reasoning about formalizing *OPT2* requirement is farfetched and is not scalable. For example, if Zoo management decides to install one more appliance for controlling Zoo entrance, and the corresponding requirements will enrich the precondition of the *enter* feature, the whole reasoning will be invalidated. The author found more scalable and intuitively plausible way to formalize this requirement in Eiffel. The corresponding formalism will be available in work [12].

IV. CONCLUSION

The specification method discussed in this work is suitable not only for formalizing statements which were also formalized in [6], but also for formalizing things which cannot be formalized with the classical tools used in [6]. Not only the requirements specification items were expressed, but also the object-oriented blueprint was built ready to equip it with code actually doing something useful. Such implementation exists and is available here: <https://github.com/anaumche/Zoo-Turnstile-Multirequirements>.

A. Pros & Cons

It is necessary to evaluate the method against the characteristics of the hypothesis introduced in section I-C:

- 1) Simultaneity of specifying the requirements and building the design: indeed, all the code fragments corresponding to different specification items merged together will bring a complete design solution available at <https://github.com/anaumche/Zoo-Turnstile-Multirequirements> (the classes ending with "_abstract").
- 2) Traceability between the specification and the implementation: the classes ending with "_concrete" located at the resource given in 1 contain the implementation and are inherited from the specification classes
- 3) Provability of the classes: this is the subject to further investigation
- 4) Continuity of the solution: since Eiffel artifacts used in the formalizations of the requirements items correspond to their natural language counterparts directly, it is visible right away how a change in one representation will affect the second one

B. Scalability

A formal representation of a requirements item specified with Eiffel is as big as the scope of the item and its natural language description are, so the overall complexity of the final document should not depend on the size of the project. Anyway, this is something to test by applying the method to a bigger project.

C. Future Work

The next steps include:

- 1) To formally prove that the specification is consistent. In particular to ensure that the features specifications preserve what is stated in the invariants; to ensure that the expressions stated in the invariants are consistent between each other: for example it should not be possible for $P(x)$ and $\neg P(x)$ to hold at the same time
- 2) To formally prove that the implementation actually satisfies the features specifications
- 3) To extend BON notation [15] so that it would be capable of expressing model-based contracts
- 4) To design machinery for translating model-based contract-oriented requirements to their natural language counterpart so that the result would be recognizable by a human being.
- 5) To apply the method to a bigger project

The AutoProof technology [14] may be utilized for automating the aforementioned proofs. AutoProof is already capable of proving that a feature implementation preserves its specification (the postcondition holds after the feature invocation assuming the precondition), and it should be empowered with the capabilities for working solely on the specifications level so that completing the goal 1 will be possible.

As a result of implementing the aforementioned plans a powerful framework for expressing all possible views on the software under construction should emerge.

ACKNOWLEDGMENT

The author would like to thank his colleagues at the Innopolis University Software Engineering Laboratory for their invaluable feedback and guidance: Dr. Bertrand Meyer, Dr. Victor Rivera, Alexander Chichigin, Dr. Manuel Mazzara.

REFERENCES

- [1] Jean-Raymond Abrial. *Modeling in Event-B: system and software engineering*. Cambridge University Press, 2010.
- [2] R Gmehlich, K Grau, M Jackson, C Jones, F Loesch, and M Mazzara. Towards a formalism-based toolkit for automotive applications. 2012.
- [3] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.
- [4] Michael Jackson. The world and the machine. In *Software Engineering, 1995. ICSE 1995. 17th International Conference on*, pages 283–283. IEEE, 1995.
- [5] Michael Jackson. *Problem frames: analysing and structuring software development problems*. Addison-Wesley, 2001.
- [6] Michael Jackson and Pamela Zave. Deriving specifications from requirements: an example. In *Proceedings of the 17th international conference on Software engineering*, pages 15–24. ACM, 1995.
- [7] Nancy G Leveson, Mats Per Erik Heimdahl, Holly Hildreth, and Jon Damon Reese. Requirements specification for process-control systems. *Software Engineering, IEEE Transactions on*, 20(9):684–707, 1994.
- [8] Dominique Méry and Neeraj Kumar Singh. Automatic code generation from event-b models. In *Proceedings of the second symposium on information and communication technology*, pages 179–188. ACM, 2011.
- [9] Bertrand Meyer. *Object-oriented software construction*, volume 2. Prentice hall New York, 1988.
- [10] Bertrand Meyer. *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.
- [11] Bertrand Meyer. Multirequirements. 2013.
- [12] Alexandr Naumchev, Bertrand Meyer, and Victor Rivera. Unifying requirements and code: an example. The work is not published.
- [13] Nadia Polikarpova. *Specified and verified reusable components*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 21939, 2014, 2014.
- [14] Julian Tschannen, Carlo A Furia, Martin Nordio, and Nadia Polikarpova. Autoproof: Auto-active functional verification of object-oriented programs. *arXiv preprint arXiv:1501.03063*, 2015.
- [15] Kim Waldén and Jean Marc Nerson. *Seamless object-oriented software architecture*. Prentice-Hall, 1995.

Intelligent Design of Class Structure Model based on Ontological Data Analysis

A.N. Kovartsev

Department of Software Systems
Samara State Aerospace University
Samara, Russia
kovr_ssau@mail.ru

V.S. Smirnov

Magistracy of the Department of Software Systems
Samara State Aerospace University
Samara, Russia
victorsmirnov92@gmail.com

S. V. Smirnov

Laboratory analysis and simulation of complex systems
Institute for the Control of Complex Systems of Russian Academy of Sciences
Samara, Russia
smirnov@iccs.ru

This paper investigates a formal approach which supports a critically significant step in object oriented analysis and software engineering. It is proposed to create an object class structure model based on an Ontological Data Analysis. Pragmatically important attributes and Ontological Data Analysis basic stages review is given.

Object-Oriented Analysis and Design; Class Structure Model; Formal Methods; Ontological Data Analysis

I. INTRODUCTION

Creating a Class Structure Model in object-oriented analysis and software engineering still remains an expert's experience realization subject. [1-7] Object and classes are the basis for the all next steps of analysis, however they "are there just for picking" (i.e. naturally appear in a statement of a problem) or are borrowed from colleagues (with or without any modification) [5]. In other words in practice there is no any systematic procedure or formalism supporting the critical for the further software engineering step.

At the same time the majority of object-oriented analysis and software engineering coryphaeus pointed out the necessity of a certain conceptual analysis of domain for "concepts" description. That is why a strict mathematical theory Formal Concept Analysis (FCA) [8] enthused object-oriented analysis and software engineering experts. Numerous researches and developments using FCA for creating Class Structure Model were accomplished. For example, [9, 10]. However, it emerged that FCA usability is limited.

- Construction of arbitrary relationships between object classes is not supported, except for the generalization relationship "is-a".
- Contradictions in the original data – a set of Basic Semantic Propositions of the form "object x has an attribute y " are prohibited. Especially the possibility of taking into account

the evidence "for" and "against" the truth of such judgments.

- Available to the designer information about the relationship of object attributes is ignored – the so-called attributes' "constraints of existence".

Although it is somewhat dampened the interest in FCA in software engineering, the method continued to develop, especially in the field of ontological modeling, for example [11, 12].

The main point of this paper is to draw developers' (especially, class structure model designers) attention to Ontological Data Analysis (ODA), the FCA evolution which can process vague and controversial data of modeled reality, discover arbitrary relationships between object classes and consider attributes' constraints of existence [13-15].

The topic of the article comes out in Fig. 1 diagram of ODA realization for Class Structure Model design.

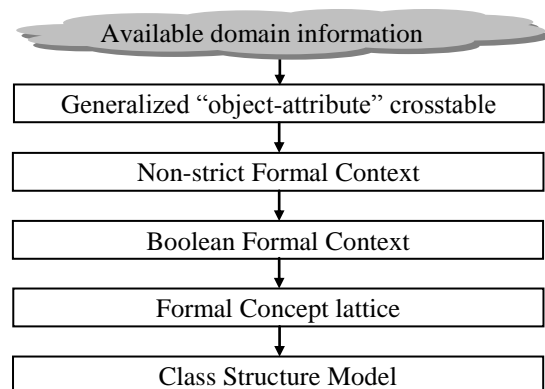


Fig. 1. Ontological Data Analysis diagram for domain Class Structure Model design

II. ONTOLOGICAL DATA ANALYSIS AND FORMAL CONCEPT ANALYSIS

ODA is a customization and a pragmatic readjustment of FCA.

For FCA primary source of initial data is a multi-valued context – “objects-attributes” crosstable (OAC) where observed domain objects’ attributes of researcher’s interest are noted.

In ODA the format of OAC is getting more complicated in order to represent domain empiric information, such as multiple independent object’s attribute records, discovering the same attribute with procedure sharing, confidence differentiation for various sources of information etc.

Besides that, as long as relations presence in ODA is treated as objects’ inner attributes demonstration, in OAC special associated attributes-valences pairs are used to represent arbitrary binary relations.

Only “weak” Basic Semantic Propositions’ estimations for domain could be extracted from such generalized OAC. These estimations form in ODA a non-strict Formal Context (FC) for conceptual framework extraction. Whereas for FCA usage a Boolean FC is necessary. Therefore ODA offers an approach for generating such FC from initial non-strict FC.

III. NON-STRICT FORMAL CONTEXT GENERATION

In OAC (general scientific form for logging empirical information) rows correspond to domain objects, columns correspond to set of objects’ attributes that are recorded by measurement procedures available to the analyst. Crosstable cells (matrix A) store the measurement results:

- set of objects $G^* = \{g_i\}_{i=1,\dots,r}$, $r = |G^*| \geq 1$,
- set of attributes $M = \{m_j\}_{j=1,\dots,s}$, $s = |M| \geq 1$,
- attributes measurement results $A = (a_{ij})_{i=1,\dots,r, j=1,\dots,s}$.

Generalized OAC is represented by tuple (G^*, M, Se, Pr, A) , where:

- $A = (a_{ij})_{i=1,\dots,m; j=1,\dots,n}$ – matrix of measurements series results Se of attributes M of objects from sample G^* , made using measurement procedures Pr . This matrix elements can be constants **NM**, **None**, **Failure** и scales characteristics of dynamic ranges of measurement procedures Pr .
- $Se = \bigcup_{i=1}^r Se_{(i)}$ – the set of all series of measurements, $|Se| = \sum_{i=1}^r |Se_{(i)}| = m$; $Se_{(i)} = \{se_{(i)k}\}_{k=1,\dots,q(i)}$, $q(i) \geq 1$, $i = 1,\dots,r$ – series of measurements, applied to object $g_i \in G^*$.
- $Pr = \bigcup_{j=1}^s Pr_{(j)}$ – arsenal of measurement procedures, $|Pr| = \sum_{j=1}^s |Pr_{(j)}| = n$; $Pr_{(j)} = \{pr_{(j)k}\}_{k=1,\dots,p(j)}$, $p(j) \geq 1$, $j = 1,\dots,s$, – set of measurement procedures used to estimate the value of the attribute $m_j \in M$, where any

procedure $pr_{(j)k}$ has a degree of confidence in its results $t_{(j)k}$.

- **None** – a result that demonstrates a finding of a measured attribute value outside of sensitivity threshold and the dynamic range of a measuring instrument; it shows a “semantic mismatch” of the object and the measuring procedure etc.
- **Failure** – a result that records measurement failure (denial, measurement means malfunction, abstention, etc.).
- **NM** (*not measured*) – a result indicating that as a matter of fact in this series of measurements corresponding property was not measured.

Non-strict FC is a tuple (G^*, M, I) , where G^* – empirical training set of domain objects, M – number of attributes of objects recorded by measuring procedures available to the researcher, I – matrix estimates all the Basic Semantic Propositions, each element b_{ij} determined in accordance with the multi-valued logic V^{TF} vector $\langle Truth, Lies \rangle$ [16]:

$$b_{ij} = \langle b_{ij}^+, b_{ij}^- \rangle; b_{ij}^+, b_{ij}^- \in [0, 1],$$

wherein component $Truth\ b_{ij}^+$ formed by evidences confirming the Basic Semantic Proposition and the component $Lies\ b_{ij}^-$ – by evidences denying it.

Building a non-strict incidence “objects-attributes” I begins with the transition from the primary data, structured in the form of a matrix A , to their semantic interpretation in the form of non-strict incidence “series-procedures” I' :

$$b_{ij} = \left\{ \begin{array}{ll} \mathbf{T} = \langle 1, 0 \rangle, & \text{if } a_{ij} \in \overline{Pr_{(j)}}; \\ \mathbf{F} = \langle 0, 1 \rangle, & \text{if } a_{ij} = \mathbf{None}; \\ \mathbf{N} = \langle 0.5, 0.5 \rangle, & \text{if } a_{ij} \in \{\mathbf{Failure}, \mathbf{NM}\}. \end{array} \right\},$$

where \mathbf{T} , \mathbf{F} and \mathbf{N} – truth constants V^{TF} logic of “True”, “False” and “Neutral” respectively, $\overline{Pr_{(j)}}$ – the set of symbols scales procedures used to measure the properties of $m_j \in M$.

Then incidence I' is transformed into a non-strict incidence “objects-attributes” I by combining the truth values of basic semantic judgments obtained for the object g_i in all series, and property m_j – all procedures (taking into account confidence in each procedure). Alignment is performed on various compositional rules V^{TF} logic [16].

IV. CREATING A BOOLEAN FORMAL CONTEXTS

Incidence “objects-attributes” I of non-strict FC can be expanded in his Boolean alpha-section, for example,

$$I = \bigcup_{\alpha^+, \alpha^- \in [0, 1]} \langle \alpha^+, \alpha^- \rangle \cdot I^{(\alpha)},$$

$$I^{(\alpha)} = (b^{(\alpha)}_{ij})_{i=1,\dots,r; j=1,\dots,s},$$

$$b^{(\alpha)}_{ij} = \left\{ \begin{array}{ll} \mathbf{True}, & \text{if } b_{ij}^+ \geq \alpha^+ \wedge b_{ij}^- \leq \alpha^-; \\ \mathbf{False} & \text{in the opposite case} \end{array} \right\},$$

wherein the alpha-section $I^{(\alpha)}$ - normal (Boolean) level corresponding vector $\alpha = \langle \alpha^+, \alpha^- \rangle$.

In practice, alpha-section $I^{(\alpha)}$ usually used as an approximation of so called « α -approximation» the original was not-strict incidence I . However, this method in the problem of forming a Boolean FC on its lack of rigor prototype is generally incorrect because the set of measured properties of M may exist a priori relationship “constraints of existence”.

Characteristic types of this kind of binary relations are considered in [17]. So a couple of properties $m_j, m_k \in M, j \neq k$ for each object data domain (and hence, for $\forall g_i \in G^*$) can be:

- inconsistent, if, possessing property m_j , object g_i obviously does not have property m_k , and vice versa;
- caused, if possessing property m_j , object g_i indisputably has the property m_k , although the reverse may be wrong;
- interdependent, if possessing property m_j , object g_i definitely has the property m_k , and vice versa.

The usual method of alpha-section is insensitive to such relations. Therefore, its application to the formation of a Boolean FC original non-strict context may lead to a violation of “constraints of existence”.

The idea of intelligent alpha-sectional non-strict FC is available for the formalization of context “constraints of existence” as a single predicate “alpha-section correctly” with argument “Threshold α of confidence in the source data” followed by the identification of the tolerance range α , delivering such a predicate **True**.

In general, set the specified area for non-strict FC is very difficult; it is possible and that it is empty. Therefore, to solve the problem correctly Boolean approximation non-strict FC in the ODA path is a reasonable compromise. Work with a common threshold of confidence α proposed to replace the manipulation of a set of thresholds of confidence in the data fragments that describe each object $g_i \in G^*$ at the level of each separately taken “constraints of existence.”

A very important case is when the inconsistency of attributes is the result of a fundamental cognitive procedure, known in FCA as a conceptual scaling [8]. This case is considered in [18], where proposed the method of rational alpha-section non-strict FC.

V. FORMATION OF CLASS STRUCTURE MODEL

Analysis of Boolean FC allows deduce all the formal domain concepts. Formal concepts are partially ordered by inclusion of extensions (the extension of the concept - a set of objects, which are described by means of this concept) and form a complete lattice [8]. To use this result in the design of the software necessary to transform formal concept lattice in Class Structure Model.

Formal concepts according to the formation of their extensions are divided into three types:

- The concepts of the first type describe objects really exist in the analyzed domain. These concepts define a

class of objects that deserve the naming of “fundamental”.

- The concepts of the second kind - only generalize other notions. In software design these classes are known as “virtual”.
- The third type of concepts is characterized by combining these features concepts first and second kinds.

When designing the Class Structure Model pragmatic considerations require confine fundamental and virtual classes of objects. In general, you can specify the following principles of formal concept lattice transformations in Class Structure Model:

- all the concepts of the lattice are candidates for fundamental classes of the model;
- the fundamental class becomes the minimum (in the terminology of lattices) concept containing the object in its extension;
- attribute is preserved to the maximum of the concepts contained this attribute in its intension;
- the highest concept lattice (his sign - power extension equal to the power set of objects) is certainly excluded from the model, if its intension is empty;
- the smallest concept lattice (his sign - the power intension equal to the power set of attributes) are known to be excluded from the model if its extension is empty;
- analysis of candidates in the fundamental classes begins with the smallest concept, and conducted by levels nearest super-concepts.

Conversion Formal Concept lattice in Class Structure Model

Step 1. The original version of the model is formed as a copy of the formal concept lattice.

Step 2. In the model is searched the greatest concept.

If the intension of this concept is empty, it is excluded from the model after the break of its ties with sub-concepts.

Step 3. In the model is searched the smallest concept.

If extension of the smallest concept is empty, then, first of all, this concept is excluded from the model after the break of its ties with super-concept, and, secondly, of his closest super-concepts formed a set of candidates in fundamental classes.

If extension of the smallest concept is not empty, then it will be one set of candidates in fundamental classes.

Step 4. Loop through a set of candidates:

- For each super-concept of the candidate under consideration excludes objects from extension that are within the extension of this candidate (the extension super-concept always not less than the extension sub-concept).
-

- In consideration of the candidate from the intension excludes any attribute that is part of the intension of at least one super-concept.
- If the candidate has no sub-concepts, it is recorded as the fundamental class. Otherwise for this candidate creates a new sub-concept, in which the extension is transferred (and only extension) of the candidate. This new sub-concept is fixed as the fundamental class of objects. The intension of such fundamental class is empty. The candidate is retained in the model as a virtual class with an empty extension.
- Promising candidates set of updated without repetition super-concepts of the current candidate

Step 5. If a set of promising candidates is not empty, then repeats Step 4.

VI. CONCLUSION

Formal Concept Analysis (FCA) has shown its benefits in many application areas – including the field of Software Engineering. Its use is especially valuable in the early stages of software development associated with the identification of a domain object types (classes) and relationships between these types.

Methodical equipment ontological analysis of the data significantly expands and strengthens these advantages:

- can deal with incomplete and contradictory information about the data domain, namely a situation is typical for the beginning of the software life cycle;
- organically describes and analyzes arbitrary relations between classes of domain;
- numerous priori known to analyst relationships between the attributes of domain are considered (actually an additional cognitive resource that did not use the classic FCA).

Finally, the arsenal includes ODA pragmatically oriented algorithm for transforming formal concept lattice model in describing the structure of the classes. Formed model differs in that only describes two kinds of classes with a fundamentally different technical realization.

ACKNOWLEDGMENT

This work was conducting research on the topic “Discovery of the intersubjective management principles based on ontological model of the situation” within the state task Institute for the Control of Complex Systems of Russian Academy of Sciences for 2013-2015, as well as public support of the Ministry of Education and Science of the Russian Federation in the framework of implementation of the Program of improving the competitiveness of Samara State Aerospace University among the world’s leading research and education centers for 2013-2020.

REFERENCES

- [1] G. Booch, *Object-Oriented Analysis and Design with Applications*. Benjamin/Cummings, 1994.
- [2] P. Coad and E. Yourdon, *Object-Oriented Analysis*. Prentice Hall, 1990.
- [3] J. Martin and J. Odell, *Object-Oriented Analysis and Design*. Prentice Hall, 1992.
- [4] S. Shlaer and S.J. Mellor, *Object Lifecycles, Modeling the World in States*. Yourdon Press, 1991.
- [5] B. Meyer, *Object oriented software construction*. 2 ed., Prentice Hall, 1997.
- [6] G.N. Kalyanov, CASE strukturnyj sistemnyj analiz (avtomatizaciya i primeneniye) [CASE structural systems analysis (automation and application)]. Lori, 1996. (In Russian).
- [7] A.M. Vendrov CASE-tehnologii: sovremennye metody i sredstva proektirovaniya informacionnykh sistem [CASE-technology: modern methods and tools for the design of information systems]. Finance and Statistics, 1998. (In Russian).
- [8] B. Ganter and R. Wille, *Formal Concept Analysis. Mathematical foundations*. Springer-Verlag, 1999.
- [9] R. Godin, H. Mili, G.W. Mineau, R. Missaoui, A. Arfi and T.-T. Chau, “Design of Class Hierarchies based on Concept (Galois) Lattices”, *Theory and Application of Object Systems (TAPOS)*, 1998, 4(2), pp. 117-134.
- [10] W. Hesse and T. Tilley “Formal Concept Analysis Used for Software Analysis and Modelling”, *Formal Concept Analysis (Foundations and Applications)*, LNAI 3626, 2005. Eds.: B. Ganter, G. Stumme, and R. Wille. Springer-Verlag, pp. 288-303.
- [11] H.-M. Haav “A Semi-automatic Method to Ontology Design by Using FCA”, *Proc. of the CLA 2004 International Workshop on Concept Lattices and their Applications* (Ostrava, Czech Republic, 2004, September 23-24). Eds.: V. Snasel, R. Belohlavek. TU of Ostrava, Dept. of Computer Science, 2004, pp. 13-24.
- [12] C. De Maio, L.V. Fenza and S. Senatore “Towards Automatic Fuzzy Ontology Generation”, In: *Proc. of the 2009 IEEE International Conference on Fuzzy Systems* (Jeju Island, Korea, 2009, August 20-24), pp. 1044–1049.
- [13] S.V. Smirnov “Onologicheskij analiz predmetnykh oblastej modelirovaniya [Ontological analysis of modeled domains]”, *Izvestiya Samarskogo nauchnogo tsentra RAN*, 2001, vol. 3(1), pp. 62-70. (In Russian).
- [14] S.V. Smirnov “Postroenie ontologii predmetnykh oblastej so strukturnymi otnocheniyami na osnove analiza formal’nykh ponyatij [Designing of ontologies by using Formal Concept Analysis in domains with arbitrary relationships]”, *Znaniya – Ontologii - Teorii: Materialy Vserossiyskoy konferentsii* (Novosibirsk, Russia, 2011, October 3-5). Vol. 2. Institut matematiki SO RAN, 2011, pp. 103-112. (In Russian).
- [15] A.N. Kovartsev, V.S. Smirnov and S.V. Smirnov “Intellectualizatsiya formirovaniya konteksta dlya vyvoda ponyatiynoi struktury predmetnoy oblasti [Intellectualization context generation to search conceptual structure of domain]”, *Informacionnye tehnologii i sistemy: Trudy 4 mezhdunarodnoy nauchnoy konferentsii* (Bannoe, Russia, 2015, February 25 – March 1). Eds.: Yu.S. Popkov, A.V. Mel’nikov. Tchelyabinskij gosudarstvennyj universitet, 2015, pp. 82-83. (In Russian).
- [16] L.V. Archinski *Vektornye logiki: osnovaniya, koncepcii, modeli* [Vector logic: foundation, concepts and models]. Irkutskij gosudarstvennyj universitet, 2007. (In Russian).
- [17] N. Lammari and E. Metais “Building and maintaining ontologies: a set of algorithms”, *Data & Knowledge Engineering*, 2004, vol. 48(2), pp. 155-176.
- [18] V.P. Ofitserov, V.S. Smirnov and S.V. Smirnov “Metod al’fa-secheniya nestrogikh formal’nykh kontekstov v analize formal’nykh ponyatij [Method alpha-section nonstrict formal contexts in Formal Concept Analysis]”, *Problemy upravleniya i modelirovaniya v slozhnykh sistemakh: Trudy XVI mezhdunarodnoy konferentsii* (Samara, Russia, 2014, June 30 – July 03). Samarskiy nauchnyj tsentr RAN, 2014, pp. 228-244. (In Russian).

Method of Symbolic Test Scenarios Automated Concretization

Nikita Voinov, Pavel Drobintsev, Alexey Veselov,
Vsevolod Kotlyarov
Saint-Petersburg State Polytechnic University
Saint-Petersburg, Russia
voinov@ics2.ecd.spbstu.ru

Alexander Kolchin
Glushkov Institute of Cybernetics NAS Ukraine
Kiev, Ukraine

Abstract— Described in the paper is an approach to symbolic test scenarios concretization in the scope of automated software verification and testing technology. Tools for automated concretization process based on user defined settings are presented.

Keywords — concretization; symbolic behavior scenario; software testing

I. INTRODUCTION

In the scope of software lifecycle the cost of software defects increases dramatically in accordance with development stage [1]. Avoiding defects on the stage of requirements gathering and detecting them on early stages of project lifecycle reduces the amount of corrections in the software and overall cost of development. This makes usage of methods and tools for model-based verification and testing extremely valuable [2,3]. However in the toolsets which mainly resolve problems of model-based approach (automation of requirements formalization, creation of behavioral models, verification of generated model-based scenarios, requirements coverage analysis [4-7]) arises the combinatorial explosion problem of possible behavioral scenarios which shall be tested [8-11].

Methods of symbolic verification are very effective to reduce the behavioral space. It is possible to specify ranges of possible parameters values in symbolic scenario. Each symbolic scenario represents a set of concrete scenarios with equivalent behavior (with same sequence of events). This means that to provide required coverage of complete model behavior it is enough to select several specific scenarios from each group of behavioral equivalence instead of having to check all possible parameters values. This allows to significantly reduce the number of scenarios covering the functionality of application in the scope of selected coverage criteria. However for code generation of executable tests only scenarios with concrete values of parameters are needed. Given that modern industrial software requires many thousands of tests with complex dependencies of parameters values it is impossible to manually count and insert appropriate concrete values based on ranges in symbolic

scenario. The concretization process shall be completely automated.

This paper describes the automated concretization process for symbolic test scenarios in the scope of VRS/TAT toolset [12] providing automated generation of test scenarios based on requirements specifications formalized with basic protocol notation [13], which is a representation of Hoare triple [14].

II. OVERALL SCHEME OF CONCRETIZATION

VRS includes symbolic trace generator STG [15] which observes the formal model behavioral space and creates traces – linear sequences of events in the model. Model states are also saved in traces. The main tool for concretization is called Trace Concretization Tool (Fig. 1). It consists of three modules – Concretizer, ValueCalculator and ConcretizationView which interact between each other.

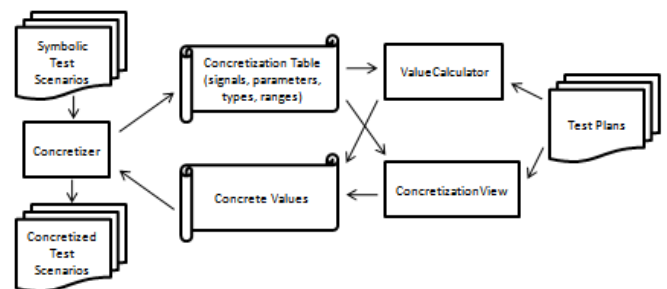


Fig. 1. Scheme of concretization

For each symbolic trace Concretizer generates concretization table with names of parameters, signals, data types and allowed ranges of values. Then while trace bypassing it calls for ValueCalculator to get concrete value for the current parameter. ValueCalculator calculates concrete value based on received commands, test plan and allowed ranges of values and returns it to Concretizer.

The implemented tools Concretizer, ValueCalculator and ConcretizationView are integrated into single concretization process which is a component of industrial software automated testing technology.

III. STEPS OF CONCRETIZATION ALGORITHM

Concretization process is iterative, on each step a single parameter is concretized. The process terminates after concretization of the last parameter in the trace.

Below some definitions are introduced. Transition in the formal model in VRS terms is a basic protocol representing parameterized transition from one model's state into another. Basic protocol $B(x)$ is represented by the following expression:

$$\forall x(\alpha(x) \rightarrow \langle P(x) \rangle \beta(x))$$

where x is a list of protocol's parameters; $\alpha(x), \beta(x)$ – a formula of basic logic language, which are called precondition and postcondition respectively; $P(x)$ – a process of basic protocol (in current case – a sequence of parameterized signals in MSC format). Trace parameters are parameters of its signals. Formula of basic language may contain variables and constants, arrays of elements of simple types, functional types. Variables which may change their values during system execution are represented by attributes and attribute expressions.

Trace is a sequence of the following type:

$$S_0 \xrightarrow{B_0(x_0)} S_1 \xrightarrow{B_1(x_1)} \dots S_n$$

where S are model's states, B – basic protocols, x – lists of their parameters.

The following steps of concretization algorithm can be specified:

- restore of initial symbolic trace
- obtain ranges of allowed values for basic protocol's parameters
- interactive concretization of trace parameters
- save concretized trace.

All steps except interactive concretization are executed automatically by internal means of VRS and hidden from outside. The most interesting for the user are implemented tools of the concretization which provide the control of concretization process and make the technology flexible enough for testing all modes of software functionality.

IV. VALUECALCULATOR TOOL

This tool implements automatic calculation of concrete values for symbolic parameters within test scenarios. One or several rules can be used for calculation: left value of the range, middle value or right value. Examples of values calculated based on ranges and selected rule are shown in the table below:

Type	Range	Rule	Calculated Value
integer	[1;9]	L	1
integer	[1;9]	M	5

integer	[1;9]	R	9
enumerated	val1,val2,val3,val4	L	Val1
enumerated	val1,val2,val3,val4	M	Val2
enumerated	val1,val2,val3,val4	R	Val4

Possible values for each parameter on each step of behavioral trace are calculated automatically by the means of VRS. Selection of the rule for value calculation is provided by corresponding set of options (Fig. 2):

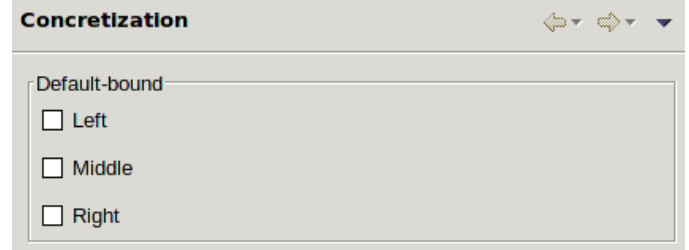


Fig. 2. Options for selecting concretization rule

Based on calculated values of symbolic parameters the STG creates traces with concrete values which can be executed on the model. When two or three rules are selected there will be two or three concretized traces generated for each symbolic scenario.

An example of tool execution is shown below. Test scenario contains a signal which turns on a radio station on the car radio. Radio station number is the signal's parameter (Fig. 3):

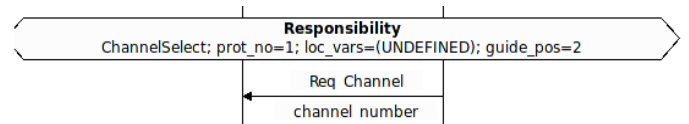


Fig. 3. A part of symbolic test scenario

If overall number of radio stations is 9, ValueCalculator will calculate the following values for the channel_number parameter depending on selected concretization rule: "1" (for the Left rule), "5" (for the Middle rule) and "9" (Right rule). If all three options are selected (Fig. 2), there will be three concretized traces generated with different values of channel_select parameter. A part of concretized trace with Right rule value selection is shown below (Fig. 4):

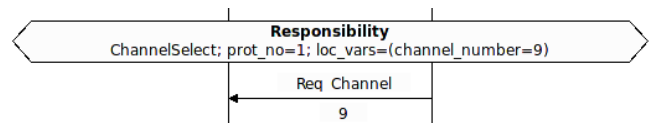


Fig. 4. A part of concretized trace with right value selected

The user can select default concretization rules and repeat generation of concretized traces with corresponding values or use ConcretizationView tool to create own test plan.

V. CONCRETIZATIONVIEW TOOL

This tool provides the ability to specify any concrete values from the possible range for one, several or all parameters in test scenario. The tool is implemented as a View element in Eclipse IDE. It allows to display the contents of concretization table and specify desired values of symbolic parameters. This is performed by adding "C" symbol on the

row with required parameter in the “Rule” column and desired value in the “Value” column.

Continue with the example of turning on a radio station of the car radio. If the range of parameter’s possible values varies between 1 and 9, then for example value 7 is neither left, nor middle, nor right value of the range. The only possible way to concretize a trace with this value is to explicitly specify it using ConcretizationView tool (Fig. 5):

ID	Parameter name	Signal name	Type	Range	Rule	Value
0	Rad	Radio.curr_vo	1	range(5)		
1	channel_number	Req_Channel	1	range(1<=channel_number&channel_number<=9)	C	7
2	channel_number	Res_Channel	1	range(1<=channel_number&channel_number<=9)		

Fig. 5. ConcretizationView user interface

As a result the concretized trace with value 7 will be generated (Fig. 6):

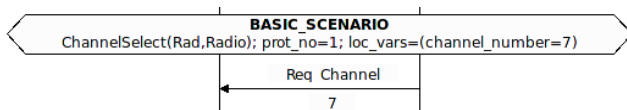


Fig. 6. A part of concretized trace with user-defined value

Applying ValueCalculator and ConcretizationView tools together the user can obtain all tests required to satisfy specific test criteria. For example, a set of tests covering all possible values of one parameter and only boundary values of another parameter. The concretization process terminates when the complete set of tests required for execution is obtained.

VI. RESULTS

Created tools were applied for preparing tests in telecom software projects. Symbolic scenarios of possible systems behaviors contained up to several hundred of basic protocols. For testing process all symbolic parameters in generated scenarios shall be concretized which is extremely time consuming without tools of automation. For example, using described approach to concretization in a small project with 11 basic protocols allowed to concretize all traces in 2 minutes. For a project with 151 basic protocols the concretization took about 20 minutes. While manual concretization of such project takes about 3 working days. Clear that in projects with several thousand of basic protocols it is impossible to concretize symbolic scenarios without automation toolset. The table below shows the comparison between manual and automated approaches to concretization:

Number of Basic Protocols in the project	Manual Concretization (staff days)	Automated Concretization (minutes)
11	0,3	2
151	3	20
464	5	25
759	8	28

VII. CONCLUSION

Integration of verification and testing allows to achieve desired level of software quality due to joining results of model static analysis after symbolic verification with number of experimental results after testing which is especially important for testing systems with wide ranges of possible values.

It is also important that symbolic scenarios can not be used for execution on the model. They shall be concretized prior to generating test code for target platform.

Implemented tools which are integrated into single chain of concretization in the scope of test automation technology [16], successfully resolve a very time-consuming problem of symbolic scenarios concretization. Also the technology allows to control coverage of boundary test parameters values which increases the quality of developed software.

References

- [1] Boehm B., Software Engineering Economics, Prentice Hall, Inc. Englewood Cliffs, New Jersey, N.Y. 1981. – 767 p.
- [2] Utting, M. and Legeard, B., Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2010.
- [3] Burdonov, I., Kosachev, A., Ponomarenko, V., and Shnitman, V., Review of Approaches to Verification of Distributed Systems, M.: ISP RAS, 2006.
- [4] TestOptimal // www.testoptimal.com
- [5] Qtronic // www.conformiq.com
- [6] Test Designer // www.smartesting.com
- [7] Spec Explorer: Microsoft Research // <http://research.microsoft.com/specexplorer>
- [8] Primeneniye metoda evristik dlya sozdaniya optimalnogo nabora testovykh stsensariyev / N. V. Voinov, V. P. Kotlyarov // Nauchno-tekhnicheskiye vedomosti Sankt-Peterburgskogo gosudarstvennogo politekhnicheskogo universiteta. Informatika. Telekommunikatsii. Upravleniye. – 2010. – T.4 – № 103. – S. 169–174.
- [9] Grindal M. Handling Combinatorial Explosion in Software Testing. Department of Computer and Information Science, Linköping universitet, 2007.
- [10] C. Nie and H. Leung, “A survey of combinatorial testing,” ACM Comput.Surv., vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011.
- [11] J. McGregor, “Testing a software product line,” in Testing Techniques in Software Engineering. Springer, 2010, vol. 6153, pp. 104–140.
- [12] Baranov S.N., Drobintsev P.D., Kotlyarov V.P., Letichevsky A.A. Implementation of an integrated verification and testing technology in telecommunication project. Proceedings // IEEE Russia Northwest Section. 110 Anniversary of Radio Invention conference. S.Petersburg, 2005. 11 p.
- [13] Letichevsky J., Kapitonova A., Letichevsky Jr., Volkov V., Baranov S., Kotlyarov V., Weigert T. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Computer Networks. 2005. 47. P. 662–675.
- [14] Hoare, C.A.R., Communicating Sequential Processes, Prentice Hall, 1985.
- [15] Letichevsky Jr., A. and Kolchin, A., Test scenarios generation based on formal model, Programming Problems, 2010, nos. 2–3, pp. 209–215.
- [16] Drobintsev P. D., Kotlyarov V. P., Nikiforov I. V., Letichevsky A. A., Incremental approach to the technology of test design for industrial projects, Modeling and Analysis of Information Systems, 2014, Volume 21, Number 6, 144–154.

Unified Model for Testing Object-Oriented Application Development Tools

Pavel P. Oleynik, PhD, System Architect Software, Aston OJSC,
Associate Professor, Shakhty Institute (branch) of Platov South Russian
State Polytechnic University (NPI), Russia, Rostov-on-Don, xsl@list.ru

Abstract— The paper presents a unified model for testing tools for object-oriented application development. Based the available papers were identified shortcomings of existing work and identified the following optimal criteria, which shall comply the resulting model:

1. To deep inheritance hierarchies
2. To presents of multiple inheritance hierarchies
3. To presents of abstract classes in the hierarchy
4. To presents of multiple (n-ary) associations
5. To presents of associations with attributes
6. To presents of a composition between classes
7. To presents of recursive associations
8. To presents of associations between classes belonging to the same inheritance hierarchy
9. To presents of association classes
10. To presents between the association class and other classes
- 11 To presents enumerations in model

With a unified graphical language UML class diagram unified model testing. The paper we verified compliance with the resulting implementation of the selected criteria was presented.

Currently the implementation of applications using object-oriented programming languages and relational databases. To overcome the object-relational mismatch it is necessary to implement object-related mapping patterns presents. The paper presents three methods used to represent the class hierarchy highlighted the advantages and disadvantages of each method.

For test the feasibility a unified model chosen development environment SharpArchitect RAD Studio which is designed object applications in C# and are implementing a relational database. The paper presents the developed object model in the form a class diagram showing the interfaces and inheritance relations diagram containing all the tables and columns the resulting database.

In the conclusion recommendations on the areas for further development work and identified the need of implement a unified model with other approaches proposed by the authors was used.

Keywords — *UML; Object modeling; Design of Information Systems; Databases; Object-oriented design; Object-Relational Mapping Patterns; Impedance Mismatch*

I. INTRODUCTION

At the moment there are many tools provide object approach to application development. Despite the existence of their own advantages and disadvantages the main goal is provide the advantages of the developer of object-oriented paradigm. The paper are describes in detail the unified model test tools development of object-oriented applications for

demonstration, graphical Unified Modeling Language which used. The practical implementation of the model is demonstrated by the use of classical methods (patterns) object-relational mapping (ORM) in the tool, developed the author. The object model is put into a relational database environment. This approach is most justified from the point of view the author, because the RDBMS is the most popular type of database management systems now.

II. DESIGN UNIFIED MODEL TESTING

When designing a unified testing model used the same approach as in the description of the design patterns in [1]. This approach is involves the description of reusable solutions widespread problems in software development without reference to particular domain. The main task of this section – is a description of the model and the structural elements (classes and associations), and not the correctness of the model and the accuracy of its fitness for a particular domain area.

Standard graphical language modeling various aspects of object systems is the language UML. This language is namely structural class diagrams will be discussed in this paper. As a result under the unified model test tools development of object-oriented applications we mean a class diagram, consisting of classes and attributes and containing common practice relationship classes.

The idea of the article is not new and there are works of similar subjects. In [2] has attempted to construct a unified model testing. However, there were no multiple (n-ary) associations and association with attributes that are an integral part of any complex information system.

In [3] presented test model to study the design of object-oriented databases. But the model is relatively simple, which is justified by its purpose. This article used dignity previously existing works and corrected drawbacks of them.

Before designing a unified model testing were nominated optimality criteria (OC) is representing the requirement of a certain structural elements in the class diagram, and which must comply with the finished implementation. Have been put forward the following requirements for the unified model test tools development of object-oriented applications:

1. Must have deep inheritance hierarchies. In realworld applications, very often there are deep hierarchy, is the

relational of inheritance and combining transitive least three classes.

2. To presents of multiple inheritance hierarchies. This will show a variety of options and modes available in the development tool.
3. To presents of abstract classes in the hierarchy. Abstract classes cannot have instances in the system and described as a container for attributes and methods used in the inherited (instantiated) classes.
4. To presents of multiple (n-ary) associations. In applications that automate realworld domains, often an association involving three or more classes. Such a relationship is called multiple or n-ary associations.
5. To presents of associations with attributes. Many domains contain attributes that do not belong to certain entities (classes), and their values appear only in the organization of associations between instances of classes. The designing unified model should have associations with attributes.
6. To presents of a composition between classes. Composition - an association between the classes which are Part and Whole. The peculiarity is that the class represents a Part can belong to only one instance of the class that represents the Whole. In this class represents the Whole manages the life cycle is a class represents a Part. When removing the Whole all Parts also deleted. This peculiarity of behavior is very important for many application domains.
7. To presents of recursive associations. Recursive call the association, the ends of which bind the same class.

These relationships allow you to implement a hierarchy of subordination.

8. To presents of associations between classes belonging to the same inheritance hierarchy. In terms of implementation is necessary to provide the implementation of the association, the edges of which are associated classes belonging to the same inheritance hierarchy, are represents the base class and the child together.
9. To presents of association classes. Association class - an association which at the same time a class. Especially the use of that class association represents a unique association, i.e. combination of instances of classes in this association is unique.
10. To associated between the association class and other classes. From a theoretical point of view, the association class is a class, so it can participate in other associations. From the point of view of the implementation of the class association presents a class that contains the attributes (fields or properties of the programming language) that refer to other classes. In turn, for the organization of the association with the class association necessary depending class to create an attribute whose type supports class association.
11. To presents enumerations in model. From a theoretical point of view, enumeration is a set of predefined constants, and the user can not extend this set by adding new values.

In accordance with the selected criteria was implemented hierarchy shown in Fig. 1.

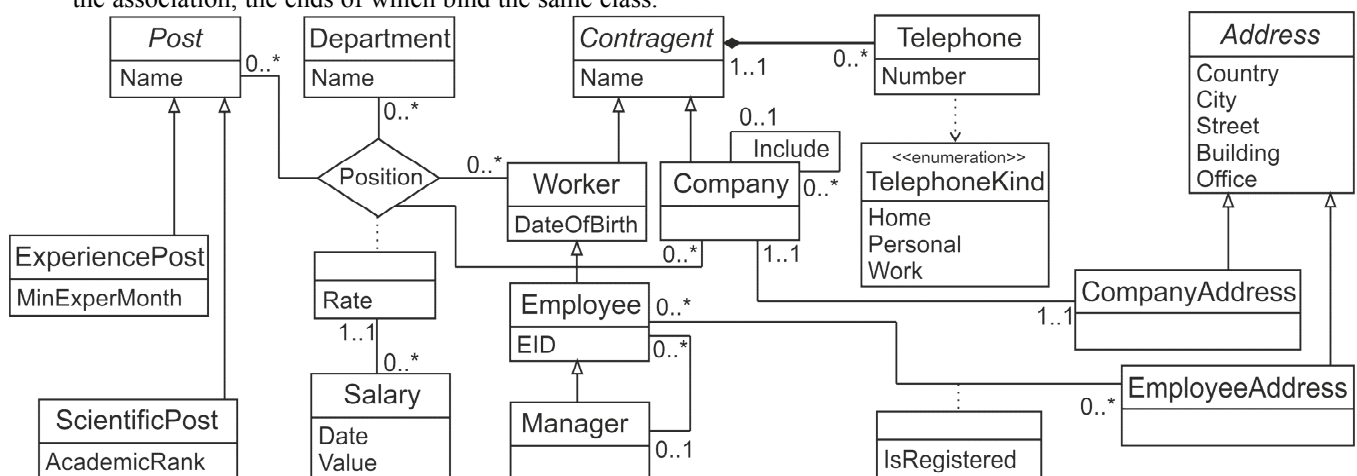


Fig. 1 - Unified model for testing object-oriented applications development tools

Consider the appointment of the main classes of diagrams are presented. As mentioned earlier this class diagram is a fictional and is not intended to describe a particular domain therefore contains some illogical (fictional) classes and associations.

For representation of employees and organizations assigned to the base abstract class Contragent. Inherited Company class is present organizations and the class Worker is the base for the employee of organization. Inherited

Employee class is an employee and an attribute EID, representing the employee unique number. Class Manager is the staff who are heads of other workers.

Post an abstract class is a position that can be occupied by staff. Inherited class ExperiencePost is a position that requires a minimum amount of experience of the applicant, expressed as number of months (attribute MinExperMonth). The second class is implemented ScientificRank describes the position of

the applicant, which requires the presence of a scientific degree, whose name is value in the attribute AcademicRank.

For presentation departments of organizations and entering into an n-ary association a class of Department was introduced. Salary class is paid wages, accrued to employees occupying positions represented by a complex association which called Position.

Class Telephone allows saving the number of phone of company. Phone type (like Home, Personal, Work) represented by enumeration TelephoneKind. For presentation address used by the base abstract class Address. Two derived class CompanyAddress and EmployeeAddress used to represent the address of the organization and address of the employee, respectively.

Check the conformity of the model presented previously selected criteria of optimality. The need for a deep class hierarchy, represented by at least three transitive inherited classes, described OC₁ and implement a class Contragent, Worker, Employee, Manager. In addition to this, there are two hierarchies: 1) Post, ExperiencePost (ScientificPost); 2) Address, CompanyAddress (EmployeeAddress). I.e. the model contains multiple inheritance hierarchies, therefore, the condition OC₂. The presence of abstract classes in the hierarchy due OC₃ and holds classes Post, Contragent and Address.

OC₄ requirements are also performed as there are n-ary association Position, combining classes Post, Department, Worker, Company. Described association has an attribute Rate, which implemented class association and binary association between Employee and EmployeeAddress classes also contains an attribute (IsRegistered) it can be argued that the requirement OC₅ fulfilled.

Each contractor represented derived from Contragent classes, a list of telephone numbers represented instances of Telephone, and both classes related with composition, OC₆ requirement is satisfied. Unified model allows you to store information about a group of companies, organize the tree structure using a recursive association connects Company class with a same. The presence of recursive association dictated OC₇.

In OC₈ written requirement for associations between classes belonging to the same inheritance hierarchy. Figure 1 between classes Employee and Manager provides this association satisfying OC₈. As previously noted, the models have a association class Position, which corresponds OC₉. Described association class is linked with addition association with Salary class. This is a consequence of the implementation OC₁₀. The presence of the models listed due to the implementation of OC₁₁. Of the present disclosure can be seen that the unified model is fully consistent with all previously selected criteria of optimality. Therefore we can move on to the implementation of the unified model.

III. THE CLASSICAL OBJECT-RELATIONAL MAPPING PATTERNS

To implement of this model development environment software systems based on the organization of the metamodel

object system presented in [4-5] was used. This development environment is called SharpArchitect RAD Studio and as storage of information uses a relational DBMS. Because information system is designed in terms of object-oriented paradigm, and implemented in a relational database environment, there is a so-called "object-relational impedance mismatch" to overcome the consequences of which object-relational mapping patterns are used. The most commonly used patterns for represent the class hierarchy.

In SharpArchitect RAD Studio implemented three classic patterns for implementing object-oriented inheritance relationships of classes in a relational structure (relational tables), presented in Fig. 2 [2, 4].

Consider the basic patterns is presented in more detail. Single Table Inheritance pattern physically represents an inheritance hierarchy of classes in a single relational database table whose columns correspond to the attributes of all classes within the hierarchy and allows you to display the structure of inheritance and to minimize the number of joins that must be performed to extract information. In this pattern each instance of the class represented by one row of the table. When you create the object values are entered only in the columns of the table that match the attributes of the class, and all the rest are empty (have a null-value).

The pattern has advantages:

- In the structure of the database contains only one table are representing all classes of whole hierarchy.
- To selection of instances of classes hierarchy do not need to make the joins of tables.
- Move fields from a base class to a derived (as well from the derivative in the base) does not require changes to the structure of the tables.

The pattern has disadvantages:

- In the study of the structure of the database tables can cause problems, because not all the columns in the table are intended to describe each domain class. This complicates the process of refining the system in the future.
- If you have a deep inheritance hierarchy with a large number of attributes, many columns can have empty values (null-values). This leads to inefficient use of the available space in the database. However, modern DBMS can compress strings containing a large number of null-values.
- Table may be too large and contain a huge number of columns. The main way to optimize the query (to reduce the execution time) is created a covering index. However, the index set and a large number of queries to a single table can lead to frequent blockages that will have a negative impact on the performance of software applications.

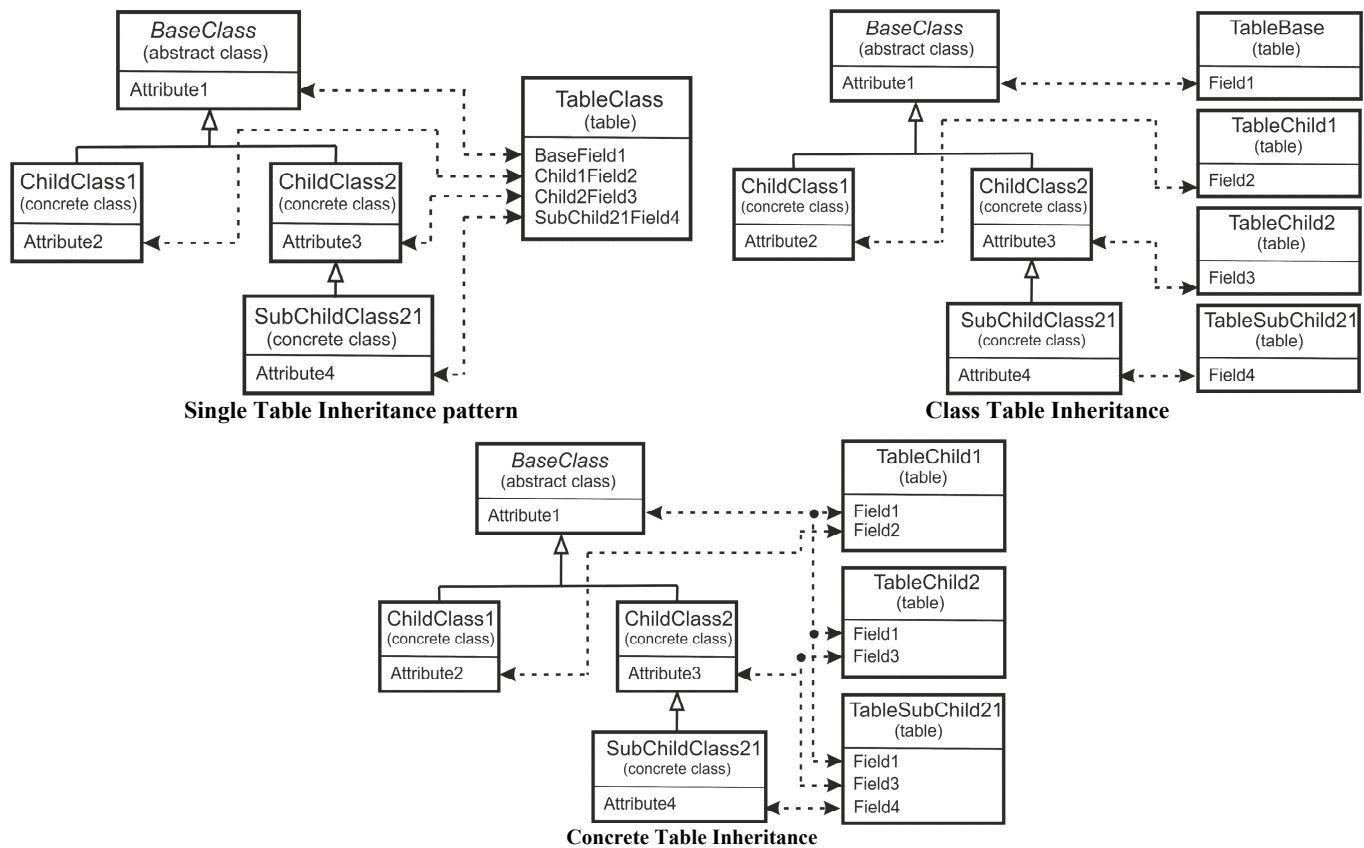


Fig. 2 - Classical object-relational mapping patterns which used to represent the class inheritance in the form of a relational structure (relational tables)

An alternative pattern is called Class Table Inheritance, representing a hierarchy of classes for one table for each class (as an abstract and concrete). Class attributes are mapped directly on the columns of the corresponding table. With this method, the key is the task of joins the respective rows of several database tables that represent a single object of domain.

The pattern has the following advantages:

- Each table contains a field, the corresponding attribute of a certain class. The therefore tables are easy to understand and take up little space on your hard drive.
- The relationship between the object model and relational database schema is simple and clear.

However, there are disadvantages:

- When you are create an instance of a particular class you want to upload data from several tables, which requires either their natural joins or a plurality of database calls followed by join results in memory.
- Move the fields in the derived class or base class requires changes in the structure of several relational tables.
- Base class table can become weaknesses in performance, since access to such tables will be carried out too often, leading to a variety of locks.

- High degree of normalization can be an obstacle to the implementation of unplanned advance queries.

The Concrete Table Inheritance pattern present is an inheritance hierarchy of classes using one table for each concrete (non-abstract) class of the hierarchy. From a practical perspective, this pattern assumes that each instance of the class (object), which is in memory, will be shown on a separate row in the table. In addition, each table in our case contains columns corresponding to attributes as a particular class, so all of his ancestors.

The advantages are that:

- Each table not contains extra fields, so that it is convenient to use in other applications that do not use object-relational mapping tools.
- When creating objects of a certain class in the application memory and retrieve data from a relational database sample is made of a single table, i.e. is not required to perform relational joins.
- Access to the table is carried out only in the case of access to a particular class, thus reducing the number of locks imposed on the table and spread the load on the system.

There are disadvantages:

- Primary keys can be inconvenient by handling.

- There is no ability to model relationships (association) between abstract classes.
- If the class attributes are moved between base classes and derived classes needed to change the structure of several tables. These changes are not as often as in the case of Class Table Inheritance pattern, but they cannot be ignored (as opposed Single Table Inheritance pattern in which these changes are absent).
- If in base class to change the definition of at least one attribute (for example, change the data type), it will require to change the structure of each table representing a derived class because a superclass fields are duplicated in all tables of its derived classes.
- In implementing the method of searching for data in the abstract class is required to view all the tables represents an instance of the derived classes. This requires a large number of database calls.

Selection of an required ORM-pattern depends on the initial logical model, i.e. from the class hierarchy of the domain. At the same time can be used two or more ORM-patterns, which is associated with the need to optimize the structure of a relational database and reduce the number of tables used, which will increase the speed of data retrieval queries.

After describing SharpArchitect RAD Studio object-relational mapping patterns which are available to the developer we can start implementing the unified model for testing tools.

IV. IMPLEMENTATION OF THE UNIFIED TESTING MODEL

In order to simplify the implementation of the three existing class hierarchies in Figure 1 will separate in available ORM-patterns. The result is shown in Fig. 3.

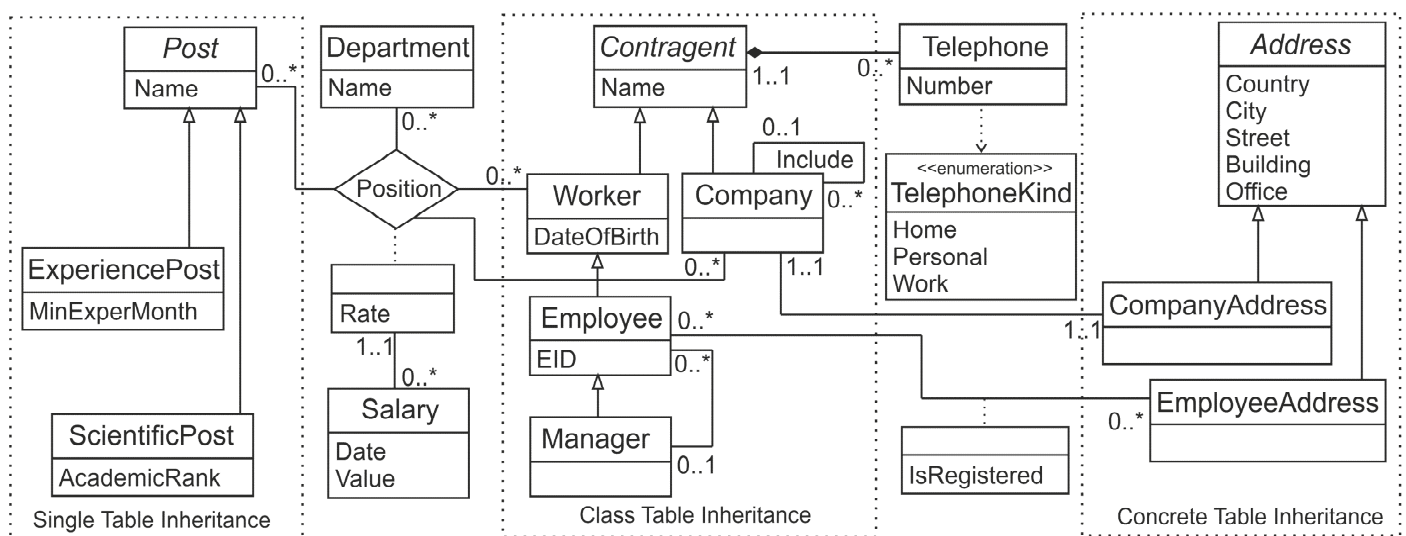


Fig. 3 - The use of the classical ORM-patterns for the implementation of the unified model for testing object-oriented applications development tools

The Single Table Inheritance for the class hierarchy Post, ExperiencePost (ScientificPost) was used. As a result, it is assumed that in the RDB will create one single table (relational table), which will be retained instances of all listed non-abstract classes. For the class hierarchy with classes Contragent, Worker (Company), Employee, Manager uses the Class Table Inheritance pattern. I.e. for all classes regardless of whether he or abstract concrete will create a separate table in RDB. Address class is abstract and has no association with other classes in model, so it will not create a separate table in the RDB. And for child classes will be created two tables (one for each heir). I.e. in hierarchy Address, CompanyAddress (EmployeeAddress) was used Concrete Table Inheritance. For other classes outside the hierarchy described, will be created on a separate relation table.

One of the main features of SharpArchitect RAD Studio support multiple inheritance is implemented by means of interfaces C# language construction, as described in detail in [4]. Used C# language does not support this syntax as an association. To represent the binary associations, regardless of

the multiplicity was used properties (property construction), containing a single value or collection of values.

Multiple n-ary association are represents a separate class, the attributes of these associations (as well as the attributes of binary associations) are converted into property of classes. To simplify information searching and extraction of all the associations are bidirectional both ends of the relevant classes there are properties whose type corresponds to the opposite end of the class association. All of the above arguments are presented graphically in Fig. 4.

In implementing the interfaces used language C#, so it is impossible italics abstract classes. Bidirectional associations are shown corresponding arrows connecting classes. In implementing the association used the following approach. From the "one" was declared property, which is a type of list (C# type IList<>), containing the elements, which is a type of class, located on the side "to-many". From the "to-many" is declared in the class property whose type is a class, located on the side "one". Association of the "many-to-many" (without attributes) can be represented by two lists is declared in class

antagonisms. In a SharpArchitect RAD Studio development environment has a number of base classes that implement the most common functionality. For example, the class IBaseRunTimeDomainClass is the root of all domain classes. To implement the tree structure will enough inherited from IBaseRunTimeTreeNodeDomainClass. At the time code generation will automatically generate additional attributes

Nodes and Owner, allow you to save a reference to the parent and subnodes, respectively. It is implemented in such a way recursive association. For submission to the transfers and sets used syntax construction "enum".

Applying the classical ORM-patterns was obtained relational database schema of the unified model now. Fig. 5 is depicts the result.

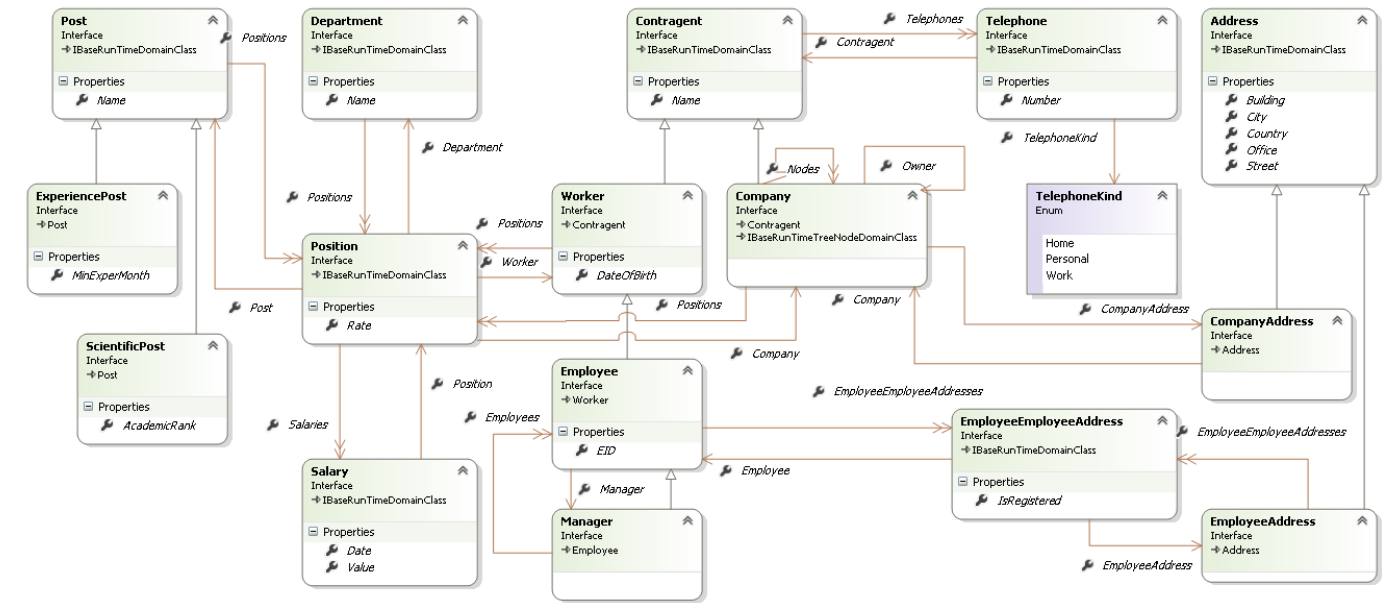


Fig. 4 - Unified model for testing object-oriented application development tools, implemented in SharpArchitect RAD Studio in C#

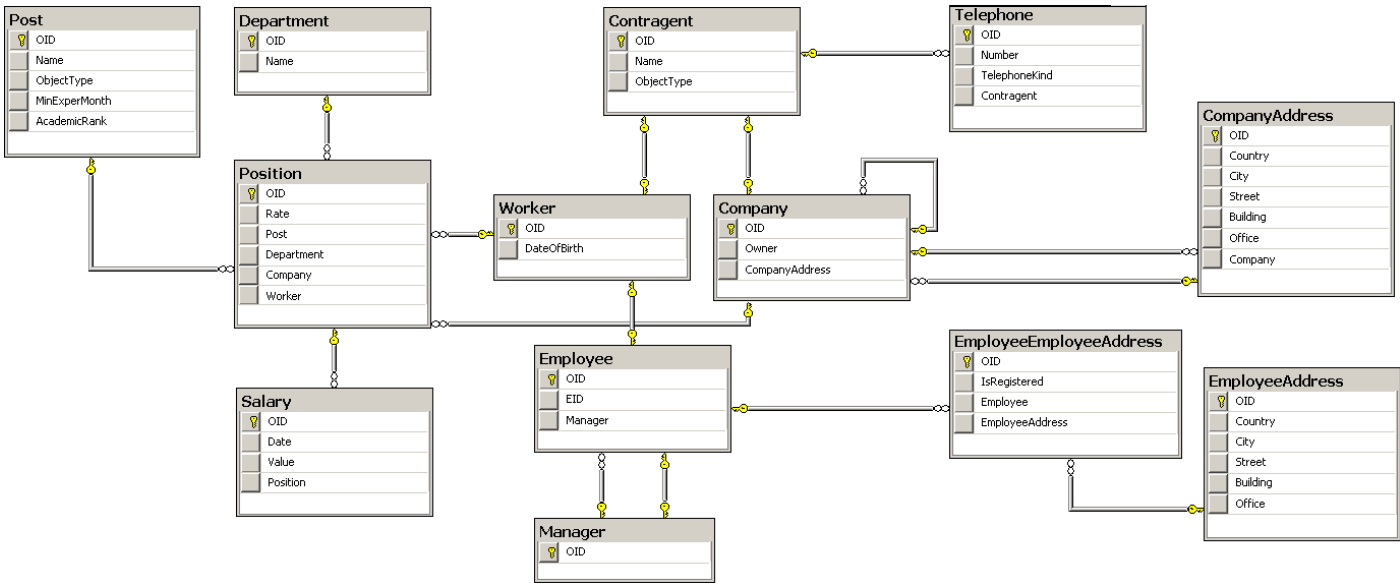


Fig. 5 - A relational database schema of the implementation of the unified model testing in SharpArchitect RAD Studio

Figure requires is explanation. For all posts submitted by three classes of Post, ExperiencePost and ScientificPost, created one single table Post, which has all the attributes of classes. Additionally, there is a column in the table OID, representing an object identifier (primary key in a relational model). ObjectType column contains the identifier of the class whose objects are stored in the form of table rows. This value

by the application to create a class of object-oriented programming language and to load the attribute values is used.

In implementing Class Table Inheritance pattern have been created for the table Contragent for abstract class and table Worker, Company, Employee, Manager for the concrete classes. Instances of classes are physically stored in multiple database tables. A copy of the Manager class is stored in all tables.

In implementing the Concrete Table Inheritance pattern is applicable for classes Address, CompanyAddress and EmployeeAddress, was created two tables: CompanyAddress and EmployeeAddress, because CompanyAddress class is abstract. All abstract class attributes stored in tables physically specific classes.

For an n-ary association Position create a separate table as well as for the binary association linking the Employee class and EmployeeAddress, for that created the table EmployeeEmployeeAddress, containing foreign keys.

Note that for the enumeration Telephone-Kind separate table is not created. An approach representations enumeration values as a bit mask and store it in the form of an integer value, where appropriate attributes are used. So the table has a column Telephone TelephoneKind, SQL-type is Integer.

After analyzing of the above it can be argued that shown in Fig. 5 implementation, created in a development environment SharpArchitect RAD Studio, fully consistent with the unified model for testing object-oriented application development tools, presented in Fig. 1.

V. CONCLUSION

Further development of the unified model is to test the feasibility of a variety of application development environments. In this alternative implementation is planned

and using the approach presented by other authors dealing with similar scientific problems.

REFERENCES

- [1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, USA, 1994, 395 pp.
- [2] Oleynik P.P. A unified model for testing object-relational mapping tools // Object Systems – 2011: Proceedings of the Third International Theoretical and Practical Conference. Rostov-on-Don, Russia, 10-12 May, 2011. Edited by Pavel P. Oleynik. - 65-69 pp. (In Russian), http://objectsystems.ru/files/Object_Systems_2011_Proceedings.pdf
- [3] Oleynik P.P. Test model for training in design of object-oriented databases // Object Systems – 2014: Proceedings of the Eighth International Theoretical and Practical Conference (Rostov-on-Don, 10-12 May, 2014) / Edited by Pavel P. Oleynik. – Russia, Rostov-on-Don: SI (b) SRSPU (NPI), 2014. – pp 86-89. (In Russian), http://objectsystems.ru/files/2014/Object_Systems_2014_Proceedings.pdf
- [4] Oleynik P.P. The Elements of Development Environment for Information Systems Based on Metamodel of Object System // Business Informatics. 2013. №4(26). – pp. 69-76. (In Russian), [http://bijournal.hse.ru/data/2014/01/16/1326593606/1BI%204\(26\)%202013.pdf](http://bijournal.hse.ru/data/2014/01/16/1326593606/1BI%204(26)%202013.pdf)
- [5] Oleynik P.P., Computer program "The Unified Environment of Rapid Development of Corporate Information Systems SharpArchitect RAD Studio", the certificate on the state registration № 2013618212/ 04 september 2013 (In Russian).

The Application of Coloured Petri Nets to Verification of Distributed Systems Specified by Message Sequence Charts*

S.A. Chernenok

A.P. Ershov Institute of Informatics Systems SB RAS
Novosibirsk, Russia
chernenksergey@gmail.com

V.A. Nepomniaschy

A.P. Ershov Institute of Informatics Systems SB RAS
Novosibirsk, Russia
vnep@iis.nsk.su

Abstract—The language of message sequence charts is a popular scenario-based specification language used to describe the interaction of components in distributed systems and communication protocols. However, the methods for validation of MSC diagrams are underdeveloped. This paper describes a method for translation of MSC diagrams into coloured Petri nets. The method is applied to the property verification of these diagrams. The considered set of elements is extended by the elements of UML sequence diagrams and compositional MSC diagrams. The properties of the resulting CPN are analyzed and verified using the known system CPN Tools and the CPN verifier based on the SPIN tool. The application of this method is illustrated with an example.

Keywords—specification; translation; verification; distributed systems; communication protocols; MSC; UML Sequence Diagrams; Compositional MSC; Coloured Petri Nets

I. INTRODUCTION

One of the major issues that arise in the process of software development is a validation problem. Over the last few years, a large number of methods and tools have been developed for the analysis and validation of systems at the stages of their design and development. However, these methods are not so powerful as compared to the formal methods of software analysis and verification. Therefore, an important goal of software validation is to improve the existing validation methods used in practice by means of integration of well-studied analysis and verification formalisms.

The scenario-based languages are a popular way to describe program specifications at the design stage of software development. They have an expressive graphical representation and are easy to use. One of the most popular scenario-based languages is the language of Message Sequence Charts (MSC) standardized by the ITU-T [14]. MSC diagrams are widely used for specification of communication protocols. The sequence diagrams of the UML standard (UML SD) [18], inspired by the MSC, made the interaction diagrams popular in the wide fields of software development. The application area of MSCs includes documentation, requirements specification, simulation, test case generation, etc.

Triggered by the increasing popularity of MSC diagrams several new dialects and extensions of the MSC language emerged. One of the important extensions increasing the expressive power of the MSC is Compositional MSC diagrams (CMSC) [9, 10]. The use of CMSC diagrams allows us to cope with the restrictions of the MSC language in order to describe a certain type of interactions, such as sliding window protocols.

It is known that at the early stages of software development the cost of errors is the highest. Therefore, the program models specified by MSCs should be valid and error-free. In practice there are tools for analysis and validation of MSC specifications. Among them are the following.

The UBET system [19, 22] can check the race conditions and timing violations for a created MSC diagram. The system also provides an automatic test case generation feature and a conversion of MSCs into the Promela language code. UBET only supports the elements of the basic MSC diagrams.

The software tools Cinderella MSC [2] and IBM Rational / Telelogic Tau [13] are visual modeling tools for analysis, specification and testing of systems described by the interaction diagrams. The system [2] supports the generation of MSC diagrams from a user application, the generation of test cases from MSCs, and the conversion of diagrams into other analysis systems. The toolkit [13] allows one to create program models based on the UML sequence diagrams, perform the automated error checking of the UML SD syntax and semantics, and to convert UML SD diagrams into the SDL modeling language for further analysis. These tools are limited by a small set of available verified properties and do not support many of the diagram elements.

The PragmaDev analyzer [8] allows one to analyze the specific properties of MSC diagrams (analysis and comparison of MSC specifications and analysis of time properties) and also some temporal logic properties defined in Property Sequence Charts. The project is under development and currently only part of MSC elements is supported.

The problem of analysis and verification of interaction diagrams is investigated by several authors.

* This work is partially supported by RFBR grant 14-07-00401

Papers [6, 7, 23] describe the modeling of UML SD diagrams using high-level Petri nets. The paper [7] deals with the translation of UML SD diagrams into CPN. This paper describes the translation rules for a limited set of diagram elements and element compositions. Also, structural restrictions are imposed on the message elements (i.e. only the synchronous messages are considered, strict sequential composition by default) and the interpretation of conditions. The paper [23] provides an extension of SD diagrams for the purpose of simulation and analysis of embedded systems. The authors describe formal translation rules for most standard elements. But the composition constructs (references) are not considered. The paper [6] provides the semantics of SD diagrams in terms of extended Petri nets. This work deals with most of the UML SD standard elements except the elements for scenario composition. Note that the translation of the elements `strict`, `break` and `critical` in these papers is not considered.

Papers [16, 20] present the translation of UML SD diagrams into the input languages of the verifiers SPIN [12] and NuSMV. The authors consider most of the diagram elements, including the combined fragments of UML SD. References and High-level MSC diagrams are not considered.

Note that most of the related work have restrictions on the diagram elements that do not allow one to specify and analyze successfully the distributed systems with independent components. In addition, these papers do not consider messages and local actions with dynamic data. The translation of CMSC diagram elements into Petri nets in the papers is not considered.

Thus, analysis and verification of MSC and UML SD diagrams is an urgent problem. Our paper is aimed at investigation of this problem.

This paper describes a method for analysis and verification of MSC diagrams of distributed systems based on the translation of diagrams into coloured Petri nets (CPN) [15]. The resulting CPN are analyzed and verified using the well-known formal methods. The choice of coloured Petri nets as a formal semantic model of interaction diagrams based on the fact that the behavioral model of CPN naturally fits the behavioral model of MSC, allowing us to simulate different types of the event composition and expressions in the MSC data language. Also, CPN are well studied and there are methods and tools for analysis and verification of net models.

The paper is organized as follows. Section 2 contains a brief description of interaction diagrams. In Section 3, a translation method from MSCs into CPN is given. Section 4 describes the translation of UML SD elements. The translation of MSC elements with data is given in Section 5. In Section 6, a translation algorithm of CMSC elements is described. Section 7 contains the size estimation of the resulting CPN generated by the translation method. The case study is described in Section 8. Section 9 contains our conclusions.

In 1992, the MSC standard [14] was developed by the ITU-T in order to obtain a simple and expressive scenario-based specification language to describe interactions in distributed systems. The significant update of the standard MSC-2000 brought new diagram elements, and the concepts of data and time. As a result, the current MSC standard can be used for description of system models at a higher level of formalization.

UML 2.0 Sequence Diagrams developed by the OMG [17, 18] are strongly inspired by the MSC. Therefore, the basic ideas, visual representation, and the set of elements in the UML SD language are very similar to MSC. The main difference is that the SD diagrams are an integral part of the UML standard. This means that all objects used in SD diagrams (processes, variables, messages, etc.) are described in various UML diagrams to detail the specific aspects of the objects behavior. On the other hand, the stand-alone MSC standard has its own syntax and can be used independently of other modeling languages in the ITU-T family. Another difference of SD diagrams is that they usually represent the control flow of an object-oriented program, whereas MSCs traditionally describe the behavior of distributed systems.

Interaction diagrams depict communication between system components (*instances*, processes, objects, etc.) by means of messages. Each diagram represents a particular scenario of the system, or a set of scenarios.

All instance events are ordered along the vertical instance axis independently of other instances. The interaction between instances is performed via *messages* which determine the relationships between events of these instances. In the MSC standard all messages are asynchronous. This means that a message output and a message input are two different asynchronous events. The UML SD standard also has a synchronous type of messages. MSCs impose a partial ordering on the set of events.

Besides the message input and output events, there are other basic MSC elements including local actions, conditions, instance creation and termination events, message gates and others [4, 5]. Also, the MSC standard provides structural elements that allow us to determine different kinds of event composition for several instances. So, MSC *inline expressions* (*combined fragments* in UML SD) provide the parallel, alternative or loop composition of events. Reference expressions and *High-level MSC diagrams* (*Interaction Overview Diagrams* in UML SD) allow us to perform the synthesis and composition of several diagrams. Note that the MSC standard defines that the connections of all structural elements within diagrams are made by means of a weak sequential composition.

Consider the example of a UML SD diagram in Fig. 1. This diagram describes the scenario of interaction between the User and Server instances. All messages except `sendData` (depicted with a message arrow of different type) are asynchronous. The operations of the user login and interaction with the server are placed in separate operands of the strict sequential composition operator `strict`, which are

separated by a dotted line. This means that further interactions with the server are impossible until all events corresponding to the user login operation are executed. After logging in, the user sends the synchronous message `sendData` and executes some local action `localWork`. After receiving message from the user, the server checks a session state. This is made in the `break` operator. If the user session has expired, the `logout` message is sent to the user and then further execution of all events within `strict` operator is terminated. Otherwise, the data transmitted to the server are stored and the user is notified about it.

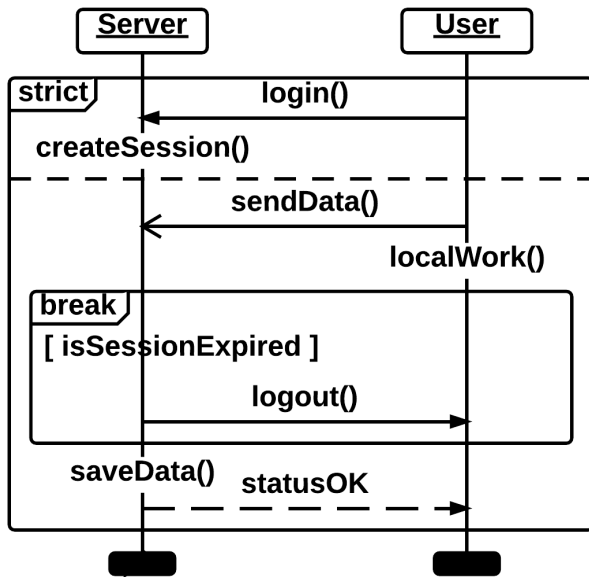


Fig. 1. An example of a UML Sequence diagram which contains the synchronous message `sendData` and two combined fragments `strict` and `break`.

III. A METHOD FOR TRANSLATION OF MSC DIAGRAMS INTO COLOURED PETRI NETS

Let us introduce the following definitions which are used in the translation algorithms of this paper.

A *structural fragment* of MSC is a subset of MSC events, which is defined by the following rules:

- a regular MSC diagram and a reference MSC diagram is a structural fragment;
- each inline expression of MSC (a combined fragment of UML SD) is a structural fragment.

Thus, an MSC diagram can be represented as a set of structural fragments connected by means of a weak sequential composition.

We define the *start events* of a structural fragment as MSC events which can be executed first among all events of this structural fragment. By analogy with start events, we also define the *final events* of a structural fragment. These are the

events that can be executed last among all events within this structural fragment.

Then, a *set of MSC traces* is a set of event execution sequences in the diagram, where each event execution sequence begins with a start event. The end of each event execution sequence can be either a final event, or an event after execution of which the MSC will not contain dynamically legal execution traces of events.

Below we present a general method to transform the MSC diagrams into CPN. The input of the translation method is an MSC, HMSC, or MSC document given in the text notation according to the MSC standard. For UML SD and CMSC elements the additional syntax is incorporated to the existing grammar of the MSC language. The output of the algorithm is a coloured Petri net in a format compatible with the CPN Tools system. In this paper we use the CPN definition given in [15]. Note that the algorithm output is a hierarchical CPN if the original specification was defined by HMSC, or if the input MSC contains MSC reference expressions.

It can be considered that the translation method has three main stages.

At the first stage an input MSC is processed to build its internal representation called a *partial order graph*. The graph is generated as follows. For each event in the MSC, a node in the partial order graph is created. This node stores some information about the event. Nodes in the generated graph are connected with each other via directed arcs. The connection between nodes is equal to the connection between the corresponding events in the input diagram.

At the second stage, processing of the partial order graph (creating auxiliary graph nodes, unfolding MSC references, etc.) is performed.

At the third stage, the partial order graph is translated into CPN. The resulting net can be described as follows. Each node of the partial order graph corresponds to a transition of CPN. Each arc connecting two nodes of the partial order graph corresponds to a place and two oriented arcs connecting two transitions of CPN. The orientation of the generated arcs in the resulting Petri net coincides with the arcs orientation in the partial order graph. The places used to transfer control between MSC events are marked by a `UNIT` colour type. The execution of an MSC event corresponds to firing of a transition in the resulting CPN. The start events of MSC correspond to the transitions with `start` input places which have an initial marking `1`()`. The final events of MSC correspond to transitions with the `end` output places and without outgoing arcs.

The translation method described above builds a CPN which simulates all possible event traces of the input MSC. In other words, the set of all possible MSC traces will coincide with the set of all possible event sequences (firing of transitions) of the resulting CPN. An initial transition of each firing sequence in the resulting CPN is a transition that corresponds to a start event of the input diagram.

Note that in this paper we do not consider the time concept of the MSC and UML SD standards. We also do not consider the following UML SD elements: *neg*, *assert*, *ignore* and *consider*. These elements do not change the set of diagram traces and hence do not affect the CPN generated by the translation method.

IV. TRANSLATION OF UML SD ELEMENTS

Since the standard of UML sequence diagrams is based on the MSC standard, most elements were adopted from MSC. In [11], the comparison of UML SD and MSC elements is made.

Several UML SD elements have different names in regard to the MSC standard terminology. For example, the instances in MSC diagrams correspond to *lifelines* in UML SD diagrams; local actions correspond to *execution occurrences*; MSC references correspond to *interaction occurrences*. In the translation algorithms described below, we will use the terminology of the MSC standard.

Note that some UML SD elements which are not in the MSC standard can be modeled by the MSC elements already discussed in [4, 5]. These elements are *continuation* (can be modeled by setting and guarding conditions of the MSC), *interaction constraint* (can be modeled by predicate conditions of the MSC), *state invariant* (can be modeled by the condition MSC element described in [4]), *conditional message* (can be modeled by a regular message within an optional operator *opt*), *operation calls / replies* (can be modeled by synchronous and asynchronous messages).

Below we consider the translation algorithms for the UML SD elements which are not modeled by the MSC elements earlier discussed.

Synchronous messages. These are the messages for which the output and input events are synchronized. This means that the sender of a synchronous message has to wait for the response from the receiver. This response will indicate what the input message processing is finished by the receiver, and the sender can continue the event execution.

The translation algorithm for the synchronous message *msg* can be described as follows. First, two transitions *Out_msg* and *In_msg* are created in the output CPN. These transitions correspond to the output and input events of *msg*. The transition *Out_msg* is connected to the transition *In_msg* via a place and directed arcs similarly to the translation rules for a regular message. Next, the transition *Reply_msg* is created which means that suspension by the process that sends the message *msg* is finished. The transitions *Out_msg* and *In_msg* are connected with the transition *Reply_msg* through the place and two directed arcs as usual.

Figure 2 shows the CPN which is the result of translation of the UML sequence diagram (see Fig. 1) with the synchronous message *sendData*.

The strict operator. This operator represents a strict sequencing between several sets of diagram events.

We define a *synchronizing event* *Es* of an MSC diagram for the instances *P1, P2, ..., Pn* ($n > 1$) as an event which can be executed only when all events from *P1, P2, ..., Pn* located before *Es* have been already executed.

The translation of the *strict* operator is performed as follows.

1. All events within the *strict* operator are translated to a CPN using the common algorithm for MSCs from Section 3.
2. For every *strict* operator with *n* ($n > 1$) operands, (*n-1*) auxiliary transitions are created in the CPN. Each created transition simulates a synchronizing event between instances involved in the *strict* operator.
3. The synchronizing transitions *Ti* ($0 < i < n$) created in the previous step are placed at the joint of *strict* operands according to the following rules. All transitions corresponding to final events of the operand *i* are connected via places to the synchronizing transition *Ti*. The synchronizing transition *Ti* is in turn connected to all transitions corresponding to start events of the operand (*i+1*). Thus, in the resulting CPN, firing of transitions corresponding to events from the operand (*i+1*) of the *strict* operator is possible only after firing of all transitions corresponding to events from the operand *i*.

A more detailed description of the translation of synchronizing events is given in [4]. Figure 2 shows the CPN which is the result of translation of the UML SD diagram (see Fig. 1) containing the *strict* operator.

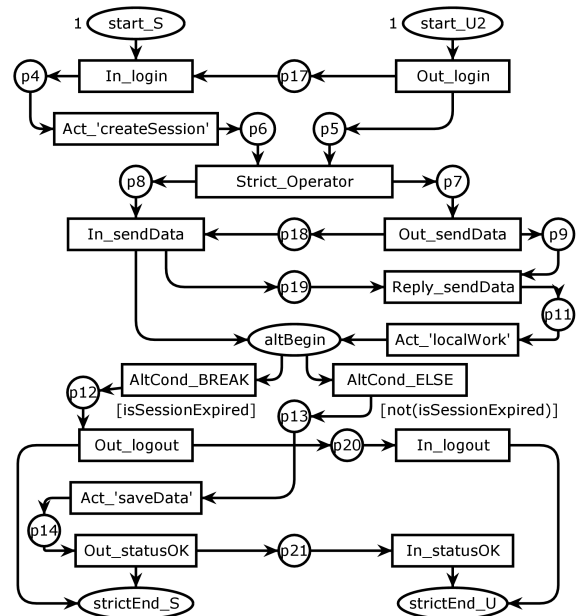


Fig. 2. CPN which is the result of translation of UML SD shown in Fig. 1.

The break operator. Semantics of this operator is similar to that of the break statement in many programming languages. If the break operator is performed in a sequence diagram, then execution of all events remaining in the enclosing (parent) structural fragment is skipped. In the UML SD standard structural fragments are called *interaction fragments*. It should be noted that the break operator is slightly different from the exceptional case operator *exc* of the MSC language [4]. In the MSC standard, the *exc* operator finishes execution of a current diagram.

The break operator belongs to combined fragments of UML SD. This fragment has one operand and should cover all instances of the parent interaction fragment. If the operand has a guard condition and the condition is true, then all events of this operand can be executed, and all remaining events of the parent fragment are ignored. If the guarding condition is false, the break operand is ignored and the rest of the enclosing interaction fragment is chosen.

The break operator can be represented as the alternative choice expression *alt* of the MSC language, where the first operand is equivalent to a single break operand, and the second operand is a part of the diagram that follows the parent fragment of the break operator.

Note that in the MSC and UML SD languages the use of the *alt* operator and its special cases (*opt*, *exc*, *break*) attached to several instances can lead to the problem of non-local choice in diagrams [1, 14, 17]. The problem is that the standards do not define which instance checks the guards, and who decides which branch should be chosen if multiple guards are true.

In our work this problem is resolved by creating the synchronizing events for each execution branch of an *alt* operator containing non-local choice. A more detailed description of the translation of an alternative expression with a non-local choice is given in [4]. The same approach is used when translating the break operator.

The translation algorithm of the break operator consists of the following steps.

1. Input and output auxiliary nodes are created for all structural fragments of a current diagram during the generation of a partial order graph.
2. Identifiers of current and parent fragments are assigned to all nodes in the partial order graph.
3. Each break fragment is translated to the output CPN according to the translation rules for *alt* operators as follows. The *alt* operator has two fixed operands. For each operand the synchronizing nodes are created to simulate a local choice. Final events of the first operand are connected to output auxiliary nodes of the parent fragment in the partial order graph (this simulates an exit from the parent fragment). Start nodes of the second operand of the *alt* operator will be output auxiliary nodes of the

break fragment (this simulates the skipping of the break operator).

Figure 2 shows the CPN which is the result of translation of the UML SD (see Fig. 1) containing the break operator.

Critical Region. The critical operator is an atomic block of events. The block atomicity is defined by two conditions. Firstly, all events within the critical region cannot be interrupted by other events of the SD diagram which are located on the same instances as this critical region. Secondly, the atomicity of events inside the critical region cannot be broken even if it is contained within the parallel execution operator *par*.

An example of the UML SD diagram containing the critical operator is shown in Fig. 3. In this diagram, when the processes *User1* and *Server* enter the critical region by the first branch of a parallel execution, the interaction with these processes in other parallel branches will not be allowed until the execution of the critical region for these processes has been finished.

To satisfy the first condition, it is necessary to create the synchronizing input and output events for each critical operator which are attached to instances involved in the critical region. The second condition is satisfied by introduction of additional places of the output CPN with flags for all events within a parent fragment *par*. Thus, an event of an instance can be executed if the flag for this instance is true. The flags for all instances involved in the critical region will be set to false when an entrance to the critical region occurs. The flags will be set to true when an exit from the critical region occurs. Note that the critical operator increases the size of the generated CPN in the case when this operator is placed to a *par*-expression with a large number of events.

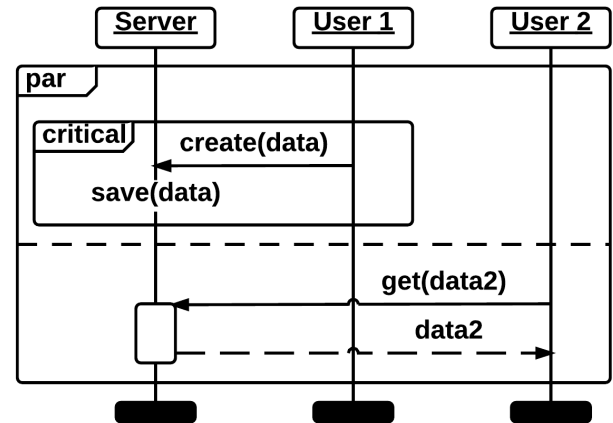


Fig. 3. An example of the UML Sequence Diagram which contains a critical region inside a *par* combined fragment.

The detailed translation algorithm of the critical region can be described as follows.

1. Synchronizing transitions are created at the beginning and end of each critical region.

2. If the critical region is not contained within a `par` operator, then the algorithm is finished.
3. If the critical region is contained within a `par` operator (if there are several nested `par` operators then we consider the highest level of nesting), then the next step is performed.
4. The fusion place *Critical* with a special colour type *CRITICALSTATE* is created. The place is defined as a CPN ML record `«record P1: BOOL * ... * Pn: BOOL»`, where *P1*, ..., *Pn* are the names of diagram instances. This place will store the information about flags for each instance, signaling about entering/finishing the critical region. The place *Critical* has an initial marking `«1' {P1=true, ..., Pn=true}»`. If a flag is true for a particular instance, this means that the instance is in a normal mode of execution. Otherwise, it is assumed that the instance has entered a critical region.
5. For each transition corresponding to an event within a higher-level `par` operator with a critical region and belonging only to instances that are involved in this critical region, the next actions are made. A bidirectional arc marked by *criticalState* (the variable *criticalState* has the colour type *CRITICALSTATE*) is created. This arc connects the place *Critical* with the current transition. The transition is marked by the CPN ML guard function `«[(#P1 criticalState) andalso ... andalso (#Pk criticalState)]»`, where *P1*, ..., *Pk* are the instance names to which the current event is attached. If the transition already has a guard function, then the above guard expression with the prefix `«andalso»` is added at the end of this function.
6. The synchronizing transition which simulates entering the critical section for the instances *P1*, ..., *Pk*, ($k \leq n$) is connected to the *Critical* place as follows. An incoming arc is marked by *criticalState*. An outgoing arc is marked by the expression `{P1=false, ..., Pk=false, ..., Pn=(#Pn criticalState)}`. This expression means that the flags of the instances involved in the critical region are reset to false, thereby preventing other events of these instances to run outside the region. This synchronizing transition is also marked by the guard function from step 5.
7. The synchronizing transition which simulates the finishing of the critical section for the instances *P1*, ..., *Pk*, ($k \leq n$) is connected to the *Critical* place as follows. An incoming arc is marked by *criticalState*. An outgoing arc is marked by the expression `{P1=true, ..., Pk=true, ..., Pn=(#Pn criticalState)}`. This expression means that the flags of the instances involved in the critical region are reset to true.

Figure 4 shows the CPN fragment which is the result of translation of the critical operator from the UML SD diagram shown in Fig. 3.

V. TRANSLATION OF DIAGRAM ELEMENTS WITH DATA

An important feature of MSC and UML SD diagrams to consider them as precise and formal specifications of software systems is the data concept.

Both standards do not impose restrictions on the data notation, so any data language can be incorporated into MSCs and UML sequence diagrams. In the MSC standard data declarations are placed in the MSC document. In the UML standard data declarations are placed in the Class Diagrams and Communication Diagrams.

In this paper we only consider the case of data declarations in the MSC document [5]. We also assume that the MSC data language allows simple types – Boolean, Integer and String – and the composite type Enumeration. An expression in the data language consists of variables, literals, parentheses, arithmetic and assignment operators, and comparisons.

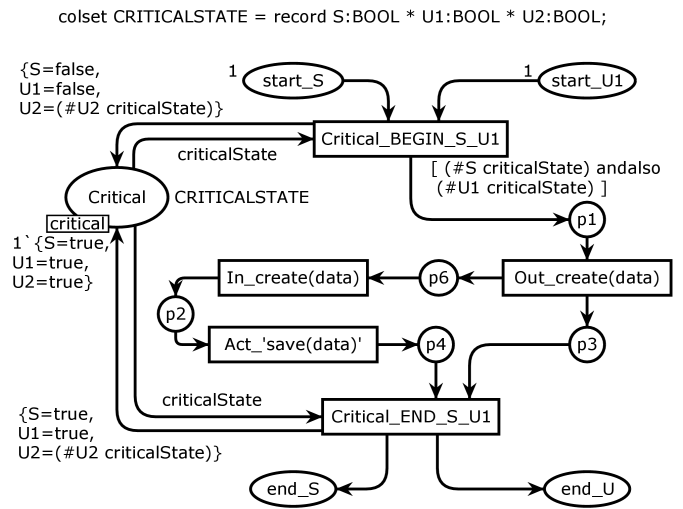


Fig. 4. The fragment of CPN which is the result of translation of critical region from the UML SD shown in Fig. 3.

The MSC document in addition to data type and variable declarations also describes the signatures of all messages with data used in the diagrams. The *message signature* $N(T_1, T_2, \dots, T_n)$ is a set of a message name *N* and the ordered set of parameter types *T_i* which defines the data tuples transmitted by this message. For example, the message signature *frame(Integer, Boolean)* means that a diagram contains a message with the name *frame*. This message transmits a data tuple with a content of Integer and Boolean types.

The data in diagrams are used in messages, local actions and conditions. Data expressions in messages and local actions can contain only variable assignment operations. A data expression in conditions cannot contain an assignment operator and can be a statement with a Boolean return value. An example of an MSC diagram containing messages with data is shown in Fig. 5.

The translation algorithm of events with data consists of two stages.

At the first stage, the colour type and variable declarations in the CPN ML language are generated from the input MSC document. These declarations will be used in the CPN obtained by translation of MSC with data events.

Generation of the colour types and variables for MSC elements with data is as follows:

- 1 Data types declared in the `data` block of the MSC document are converted into the corresponding colour types of CPN ML.
- 2 Local variables declared for each instance in the `inst` block of the MSC document are converted to variables of CPN ML with the same name and with the colour type resulting from the transformation at step 1.
- 3 Message signatures declared in the `msg` block of the MSC document are used to simulate message buffers in the resulting CPN. The signature $N(T1, T2, \dots, Tn)$ is translated to a product colour type of the CPN ML language: $colset\ pT1T2\dots Tn = product\ T1 * T2 * \dots * Tn$. To simulate the buffer which contains messages with the same signature $N(T1, T2, \dots, Tk)$, the list colour type is used: $pT1T2\dots TkList = list\ pT1T2\dots Tk$.
- 4 For colour types generated at step 3, auxiliary variables $pT1T2\dots Tn_var$ and $pT1T2\dots TnList_var$ of types $pT1T2\dots Tn$ and $pT1T2\dots TnList$ are created.

At the second stage, the translation of an MSC diagram which uses data declared in the MSC document is performed.

The translation of local actions and conditions with data is described in [5]. Below we describe the translation of messages with data. The MSC and UML SD standards imply that communicating instances send messages through the buffer which is local regarding to messages. This means that there is one FIFO buffer for every message in a diagram. Buffers which contain MSC messages with data are modeled by places of the list colour type in the resulting CPN. The list is a queue of records (CPN product types), where each record contains the set of transmitted data values. Thus, the translation algorithm for messages with data is as follows:

1. For each message $msg_i(T1, T2, \dots, Tn)$ in the diagram, a place in the resulting CPN is created to simulate the message buffer as follows. The name msg_i and the colour type $pT1T2\dots TnList$ are assigned to the place. The initial marking for this place is set up to the value $1'[]$, which indicates that the buffer is empty.
2. The input and output events of the message msg_i are translated into the corresponding transitions of the CPN.
3. Each transition corresponding to the input/output events of the message msg_i is connected to fusion places modeling the variable states. The details of variable state simulation in the resulting CPN are given in [5].

4. For a transition corresponding to an output event of the message msg_i , an input arc from the place msg_i is created with the inscription $pT1T2\dots TnList_var$. Also the output arc is created with the inscription $pT1T2\dots TnList_var \wedge [(VarT1, VarT2, \dots, VarTn)]$, where $VarTi$ are the variable names with data transmitted from the sender instance. This expression describes the addition of a tuple with a message content into the buffer.
5. For a transition corresponding to an input event of the message msg_i , an input arc from the place msg_i is created with the inscription $\langle pT1T2\dots Tn_var :: pT1T2\dots TnList_var \rangle$. This expression means that a head element and a tail part of the buffer are got and saved to the specified variables. Also, the output arc is created for this transition with the inscription $pT1T2\dots TnList_var$, which is used to simulate the removal of the upper buffer element.
6. The process of obtaining and saving the transmitted data by the receiver instance is modeled in the resulting CPN as follows. The fusion places are created for each variable listed in the actual parameters of the message signature msg_i . These places are used to store the transmitted data of the message msg_i into the local variables of the receiver process (see the translation of local actions with the data for full details [5]). The transition corresponding to the input event of the message msg_i is connected to the created fusion places. The outgoing arcs from each fusion place are marked by the corresponding variable names. The arcs coming into the fusion places are marked by the inscription $\langle Tj_var = \#j\ pT1T2\dots Tn_var \rangle$, where Tj_var is the j -th variable name of the receiver in the signature msg_i , and the expression $\langle \#j\ pT1T2\dots Tn_var \rangle$ means that the j -th element from the tuple variable $pT1T2\dots Tn_var$ is got.

Figure 6 shows the CPN which is the result of translation of the MSC from Fig. 5 containing non-regular messages with data introduced in the next section.

VI. TRANSLATION OF COMPOSITIONAL MSC ELEMENTS

The non-standard extension of MSC diagrams called *Compositional Message Sequence Charts (CMSCs)* [9, 10] has been developed to increase the expressive power of the MSC language and to describe scenarios with complex parallel communication of processes.

In [9, 10], the authors show that the expressiveness of MSC diagrams is not sufficient for the specification of a certain type of interactions, such as sliding window protocols. In the CMSC language it is possible to describe this kind of protocols using partial-defined messages. The use of this type of messages, on the one hand, allows messages to be decomposed into several diagrams. On the other hand, such messages use a different buffer type which is similar to the buffer model in the communicating finite-state machines or SDL language.

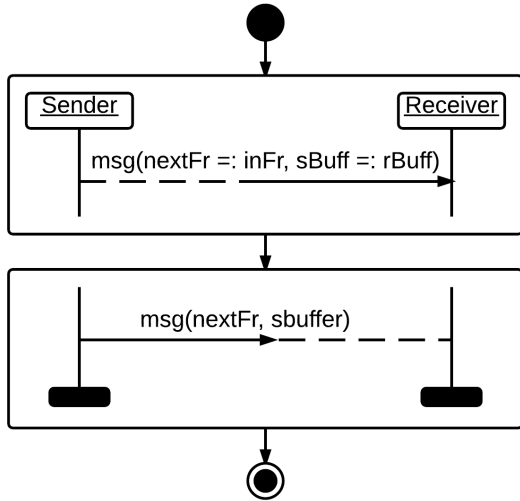


Fig. 5. The HMSC diagram with two MSCs which contain the unmatched message msg.

The CMSC language is defined as the MSC language, except for the definition of messages. In Compositional MSC diagrams, the input and output message events are partially defined. This means that for the partial-defined message there are multiple input events for a single output event and vice versa. Such messages in a CMSC are called *unmatched messages*.

Unmatched send message events and unmatched receive message events use a new buffer model. This buffer is local relative to the two instances involved in the message exchange (this is a so-called pair buffer).

An example of the CMSC diagram is shown in Fig. 5. Unmatched messages are shown as arrows with a dotted part. The CMSC shows the decomposition of the unmatched message msg which is contained in two different reference MSC diagrams.

Below we describe the translation algorithm for unmatched messages.

- 1 Each input and output event of the unmatched message $umsg_i(T1, T2, \dots, Tn)$ is converted to the corresponding transition of the CPN.
- 2 If the message does not contain any data then the following steps are made.
 - 2.1 The fusion place simulating a buffer is created with the UNIT colour type and the name «CMSC $P1$ -to- $P2$ », where $P1$ is the name of the instance that sends the message $umsg_i$ and $P2$ is the name of the instance that receives this message. Note that the name of the created place is unique for the couple of instances $P1$ and $P2$ which communicate in the direction from the first to the second instance.

- 2.2 For each transition corresponding to the output unmatched message event from $P1$ to $P2$, an output arc is created. This arc is connected to the place «CMSC $P1$ -to- $P2$ ».

- 2.3 For each transition corresponding to the input unmatched message event from $P1$ to $P2$, an input arc is created. This arc connects the place «CMSC $P1$ -to- $P2$ » with the current transition.

- 3 If the message contains data then the following steps are made.

- 3.1 The fusion place simulating a buffer is created as follows. The place type is set to $pT1T2...TnList$. The place name is set to «CMSC $P1$ -to- $P2$ - $umsg_i$ », where $P1$ is the name of the instance that sent the message with data, $P2$ is the name of the instance that receives this message, and $umsg_i$ is the message name. The place is marked by 1'[]]. Note that the name of the created place is unique for the couple of instances $P1$ and $P2$ with a given type of the message signature. Thus, the unmatched messages with the same signature will be sent by $P1$ through a common buffer. The same is true for the receiving of unmatched messages.

- 3.2 The processing of transitions corresponding to the output events of unmatched messages with data is carried out by the translation rules of step 4 of the previous section.

- 3.3 The processing of transitions corresponding to the input events of unmatched messages with data is carried out by the translation rules of steps 5 and 6 of the previous section.

Figure 6 shows the CPN which is the result of translation of the CMSC (see Fig. 5) with the unmatched message msg.

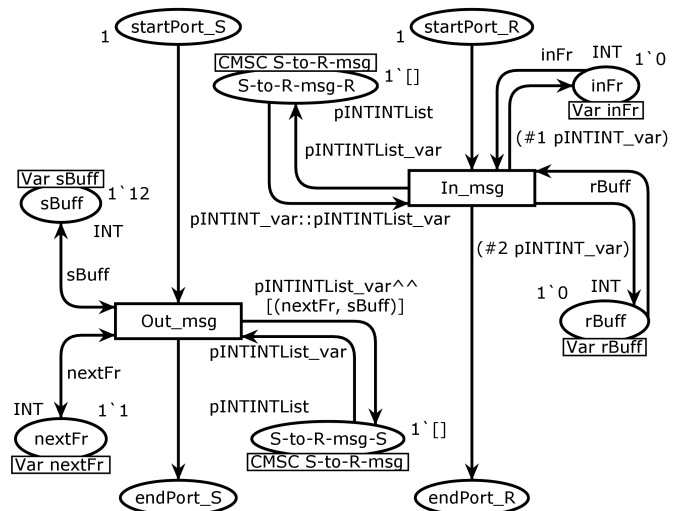


Fig. 6. The CPN which is the result of translation of the HMSC shown in Fig. 5.

VII. SIZE ESTIMATE OF THE RESULTING CPN

Below we consider the estimate of the number of transitions, places, and arcs in the CPN, given as the result of translation algorithms described in our paper.

Let us consider the MSC diagram with N events, M messages and the number P of instances containing events.

Introduce the following notation: S is the number of start and final MSC events; AC is the number of local actions and conditions; IP is the number of parallel operators `par`; IL is the number of `loop` operators; N_{IP} is the maximum number of events among `par` operators of the diagram; BR is the number of `break` operators; ST is the number of `strict` operators; OP_{ST} is the maximum number of operands among `strict` operators of the diagram; CR is the number of `critical` operators within `par` operators; VAR is the number of variables defined in the MSC.

Then the upper bound T of the number of transitions in the resulting CPN will be:

$$T \leq N + 2P \cdot (IP + IL) + ST \cdot (OP_{ST} - 1) + P \cdot BR + 2CR.$$

The upper bound P of the number of places in the resulting CPN has the following form:

$$P \leq N + M + S + VAR + 2P \cdot (IP + IL) + ST \cdot (OP_{ST} - 1) \cdot P + P \cdot BR + 2CR.$$

The approximal upper bound A of the number of arc in the resulting CPN has the following form:

$$A \leq 2N + 4M + 2 \cdot VAR \cdot (AC + 2M) + 4P \cdot (IP + IL) + 2ST \cdot (OP_{ST} - 1) \cdot P + 2P \cdot BR + 2CR \cdot N_{IP}.$$

As we can see, a significant contribution to the size estimate of the resulting CPN is made by the operators `par`, `loop`, `break` and `critical`.

VIII. CASE STUDY: ALTERNATING BIT PROTOCOL

Let us consider an example of the property verification for the MSC specification of a protocol known as the *Alternating Bit Protocol* (ABP).

This protocol is bidirectional. This means that the data between the two communicating machines are transmitted in both directions. The protocol operates as follows. The sender sends a sequence of data frames to the receiver. Each data frame consists of two parts: a one-bit frame number and a portion of data. When a data frame arrives to the receiver, it sends to the sender an acknowledgment frame that contains the number of the received frame. Both processes use a timer to wait for the next frame. Thus, the sender is sending a current data frame continuously until it receives an acknowledgment from the receiver with the current frame number. On the other hand, after getting a data frame, the receiver is sending an acknowledgment frame to the sender continuously until it receives a new data frame from the sender.

The MSC specification of the ABP protocol is presented in [3]. In the specification, the `par` operator and CMSC elements are used to describe the distributed interaction between two machines. The timer execution events of communicating processes are modeled in the resulting CPN by firing of transitions corresponding to these timer events. The transmitted data in the protocol are a sequence of integers from 1 to 4.

To reduce the state space of the resulting CPN and apply the CPN verifier based on SPIN [21], the initial MSC specification should be rewritten into a quasi-regular form in which diagrams do not contain unlimited loops [5]. To do this, we introduced additional restrictions on the protocol model without loss of generality: the frame number that can be lost during transmission is limited by a constant.

For analysis and verification of the ABP model, the following properties of a proper behavior are formulated:

1. The sequence of the received data is equal to the sequence of the sent data.
2. The receiver does not accept the same message twice.
3. The sender does not send a new message before a previous one was acknowledged.
4. The sequence of the received frames is a prefix of the sequence of the sent frames.

The property 1 is a postcondition. For the protocol model, it means that if the event execution of the MSC specification ends at its endpoint, then this property is satisfied. For the CPN model of the protocol, it means that the resulting net should not have dead markings except the markings corresponding to the endpoint of the MSC specification. Properties 2, 3 and 4 are specified by linear temporal logic (LTL) formulas [3].

The analysis of the model properties was made in the CPN Tools (property 1) and in the automated verification system developed in IIS SB RAS on the basis of SPIN (properties 2, 3 and 4). Verification of the properties described above showed that they are satisfied for the ABP protocol model.

The property validation was also made for the ABP protocol model containing errors. In the first case, we considered a protocol model in which one of the processes can send a new message non-deterministically, without waiting for reception of the previous one. In the second case, we considered a protocol model in which the sender can send non-deterministically a frame with incorrect data. During verification of these ABP models, the following property violations were detected. In the first case, property 3 was violated (and property 4, consequently). In the second case, property 4 was violated.

For the violated properties, the counterexamples were generated which contain traces in the MSC specification leading to a broken state. The file with a counterexample is a sequence of CPN transitions and net markings.

Using the counterexamples, the errors were localized in the original MSC specification. Since each transition corresponds

to a concrete event in an MSC, and the MSC variables state is calculated by the values of places with the same name as original variables, the localization of errors in a diagram by a counterexample is straightforward.

IX. CONCLUSION

The scenario-based specification languages are a convenient and expressive way to describe a system behavior during the design and development stages. The most popular in practice among them are the MSC and UML SD languages. Despite a wide application of these notations, the methods of analysis and verification are still underdeveloped.

In this paper we describe the method for translation of MSC diagrams into coloured Petri nets. To the best of our knowledge, our method is the first to cover a large set of the MSC and UML SD diagram elements with minimal restrictions on the considered elements. Unlike the related papers, the translation method fully supports the diagram elements with dynamic data and elements of compositional MSC diagrams. The consideration of all elements listed above, on the one hand, allows us to apply the translation method for most interaction diagrams used in practice. On the other hand, this allows us to use the method for verification of distributed systems with complex object interactions.

A CPN given as a result of the translation method can be analyzed and verified by the known verification methods and program tools. In particular, one can analyze some properties of MSC diagrams using the CPNTools [15], and verify properties specified by LTL formulas using the method [21].

The software tool was implemented on the basis of the translation algorithms. The translator has been tested on various examples of communication protocols. In particular, the alternating bit protocol specified by MSCs has been considered. For the protocol, the CPN model was generated. Some properties of the resulting CPN was analyzed by the CPN Tools and verified by the CPN verifier [21].

In our further work we plan to develop the approach for formal justification of correctness of the translation algorithms. We will study other MSC extensions intended for specification of distributed systems. Also, we plan to use the translator for verification of other examples of distributed systems and communication protocols.

REFERENCES

- [1] Abdallah R., Gotlieb A., Helouet L., Jard C. Scenario Realizability with Constraint Optimization // FASE 2013, LNCS 7793. — 2013. — P. 194-209.
- [2] Cinderella MSC tool // <http://www.cinderella.dk/msc.htm>
- [3] Chernenok S. A. Analysis and Verification of Message Sequence Charts of Distributed Systems Using Coloured Petri Nets. Appendix. <http://bitbucket.org/chenenok/msc-verification>
- [4] Chernenok S.A., Nepomniaschy V.A. Analysis of Message Sequence Charts of Distributed Systems Using Coloured Petri Nets // Preprint 171. — Institute of Informatics Systems SB RAS. — Novosibirsk. — 2013 [in Russian]. <http://www.iis.nsk.su/files/preprints/171.pdf>
- [5] Chernenok S.A., Nepomniaschy V.A. Analysis and Verification of Message Sequence Charts of Distributed Systems with the Help of

- Coloured Petri Nets // Modeling and Analysis of Information Systems. — 2014. — V. 21. — N. 6. — P. 94-106 [in Russian].
- [6] Eichner C., Fleischhack H., Meyer R., Schrimpf U., Stehno S. Compositional Semantics for UML 2.0 Sequence Diagrams Using Petri Nets // SDL-Forum 2005, LNCS 3530. — 2005. — P. 133-148.
- [7] Fernandes J.M., Tjell S., Jorgensen J.B., Ribeiro O. Designing Tool Support for Translating Use Cases and UML 2.0 Sequence Diagrams into a Coloured Petri Net // SCESM '07: Proceedings of the Sixth International Workshop on Scenarios and State Machines. — Washington, DC, USA, 2007. — P. 2.
- [8] Gaudin E., Brunel E. Property Verification with MSC // SDL 2013, LNCS 7916. — 2013. — P. 19-35.
- [9] Genest B. Compositional Message Sequence Charts (CMSCs) Are Better to Implement Than MSCs // TACAS 2005, LNCS 3440. — 2005. — P. 429-444.
- [10] Genest B., Muscholl A., Peled D. Message Sequence Charts // Lectures on Concurrency and Petri Nets, LNCS 3098. — 2003. — P. 537-558.
- [11] Haugen O. Comparing UML 2.0 Interactions and MSC-2000 // System Analysis and Modeling, LNCS 3319. — 2005. — P. 65-79.
- [12] Holzmann G. The Spin model checker: primer and reference manual. — Addison Wesley, 2003.
- [13] IBM Rational Tau // www.ibm.com/software/products/en/ratitau
- [14] ITU-T Recommendation Z.120 (02/2011): Message Sequence Charts (MSC). — 2011.
- [15] Jensen K., Kristensen L.M. Coloured Petri Nets: Modeling and Validation of Concurrent Systems. — Springer, 2009.
- [16] Lima V., Talhi C., Mouheb D., Debbabi M., Wang L., Pourzandi M. Formal verification and validation of UML 2.0 Sequence Diagrams using source and destination of messages // Electron. Notes Theor. Comput. Sci., 2009. — V. 254. — P. 143-160.
- [17] Micskei Z., Waeselynck H. The many meanings of UML 2 Sequence Diagrams: a survey Software and Systems Modeling. — Springer, 2011. — V. 10. — N. 4. — P. 489-514.
- [18] OMG UML 2.5, 2013 // <http://www.omg.org/spec/UML/2.5/Beta2/>
- [19] Rajeev Alur, Holzmann G.J., Peled D.: An Analyzer for Message Sequence Charts // TACAS 96, LNCS 1055. — 1996. — P. 35-48.
- [20] Shen H., Robinson M., Niu J. Model Checking Combined Fragments of Sequence Diagrams // Software and Data Technologies. — Springer, 2013. — V. 411. — P. 96-111.
- [21] Stenenko A.A., Nepomniaschy V.A. Model Checking Approach to Verification of Coloured Petri Nets // Preprint 178. — Institute of Informatics Systems SB RAS. — Novosibirsk. — 2015 [in Russian]. <http://www.iis.nsk.su/files/preprints/178.pdf>
- [22] UBET (MSC/POGA) computer toolset // <http://cm.bell-labs.com/cm/cs/what/ubet/index.html>
- [23] Yang N., Yu H., Sun H., Qian Z. Modeling UML sequence diagrams using extended Petri nets // Telecommunication Systems. — Springer, 2012. — V. 51. — N. 2-3. — P. 147-158.

Carassius: A Simple Process Model Editor

Natalia M. Nikitina, Alexey A. Mitsyuk

Laboratory of Process-Aware Information Systems

National Research University Higher School of Economics

3 Kochnovsky Proezd, Moscow, Russia

Email: nmnikitina@edu.hse.ru, amitsyuk@hse.ru

Abstract—Process models and graphs are commonly used for modeling and visualization of process models. They may represent sets of objects or events linked with each other in some way. Wide use of models in such languages engenders necessity of tools for creating and editing them.

This paper describes the model editor which allows for dealing with classical graphs, Petri nets, and finite-state machines. Additionally, the tool has a list of features like simulation of Petri nets, import and export of models in different storage formats.

In the paper one can find a detailed description of a couple of layout algorithms which can be used to visualize graphs.

Index Terms—graph, Petri net, finite-state machine, process model, process model visualization, process model editor

I. INTRODUCTION

The modern world is full of information systems working in different business domains. One of the most developed concept is *process-aware information systems* [1]. A wide variety of different notations has been developed to model processes.

In this paper we present a new tool for editing and simulating process models in different notations. Our goal is not to build yet another complicated model simulator.

Our ambition was to develop a model editor which may be used for educational purposes. Thus, the decision was made to implement a *simple* and *extensible* model editor for different modelling notations. In particular, a modular architecture of *Carassius* allowed us to implement simulation modules in addition to different editors.

The remainder of this work is organized as follows. Section II gives a description of the tool, implemented approaches and algorithms. Furthermore, the description of the tool's features is provided.

In section III we consider other tools with similar functionality. The advantages and disadvantages of these tools are provided. Section IV concludes the paper.

II. TOOL OVERVIEW

A. Functionality

Here one can see the brief description of all features implemented in *Carassius*.

In this paper we present a tool which intended to help researchers and other people easily make and edit models of different types. *Carassius* works with graphs of 3 types: classical graphs, Petri nets and finite-state machines. First of all, it permits to edit process models by hand. Besides, the tool supports several markup languages (PNML [2], [3], GraphML [4], [5] and FSAML) and can read and save models

from and into these formats. FSAML is a new XML format we developed for storing a finite state machines system.

The working area has a grid helping users position the nodes. The tool can automatically arrange model elements according to the grid. Users may set or change all the possible properties of the whole model or its parts (for example: node names, arc weights, etc). The tool can arrange models using different layout algorithms: for graphs and finite-state machines it uses the force-directed algorithm, whereas for Petri nets it uses the layering algorithm developed for *Carassius*. Both of them are described in details in subsection Visualization refinement.

In addition, *Carassius* has features for a Petri net simulation. The tool supports step-by-step *token-game* of a process model [6]. Moreover, there is a special *colouring mode* that shows the real way of tokens during the simulation. Because of these features, the tool can be used successfully in educational purposes.

B. Supported Notations

This part describes the modelling notations supported by *Carassius*.

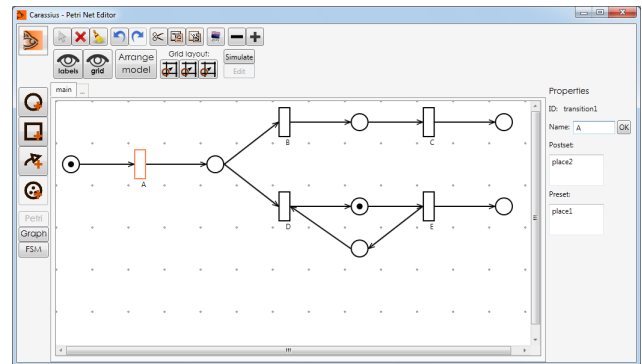


Fig. 1. A Petri net editing

Petri nets: The main supported formalism is *Petri nets*. Petri nets are widely used in process modelling [7], [6]. A Petri net is a directed bipartite graph with two types of nodes: transitions (denoted by rectangles) and places (denoted by circles). There are directed arcs between places and transitions (denoted by arrows). Places can contain so-called tokens inside, which determine the current state of a net and its

marking. Petri nets offer a graphical notation for step-by-step processes that include choice, iteration, and concurrent execution. Execution of a process is depicted by tokens flow.

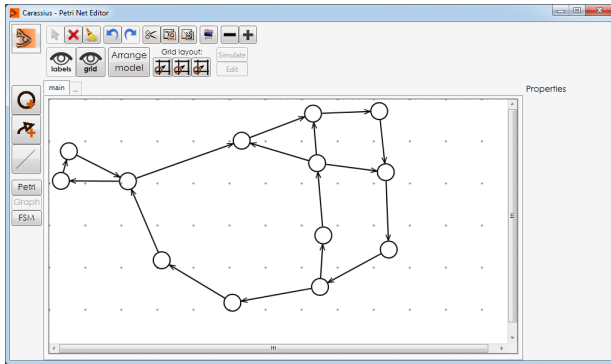


Fig. 2. A directed graph editing

Graphs: Carassius is also works with classical graphs. Both directed and undirected edges are supported. It is possible to assign weights of edges. Process of graph editing is quite simple. However, a possibility to deal with directed graphs and store them using GraphML format is very useful.

Finite-State Machines: A finite-state machine (FSM, finite-state automaton [8]) is an abstract machine that can be in an only one of a finite number of states at a point of time. FSM recognizes or accepts certain word of some language with finite alphabet. It can move from one state to another by triggering a transition with the same label as a next letter of an input word. If a FSM stops in a state from the set of so-called acceptance states, then it accepts a word. This is not always the case. Therefore, any FSM forms a language consisting of the words accepted by this FSM.

A particular FSM is defined by a list of its states and transitions. States are usually depicted by circles, and transitions are depicted by labelled directed arcs. There are two special types of states: a single starting state and a set of final (accepting) states. A starting state is depicted by a circle with an arrow from anywhere going into the circle (see figure 3). Each accepting states is depicted by a double circle.

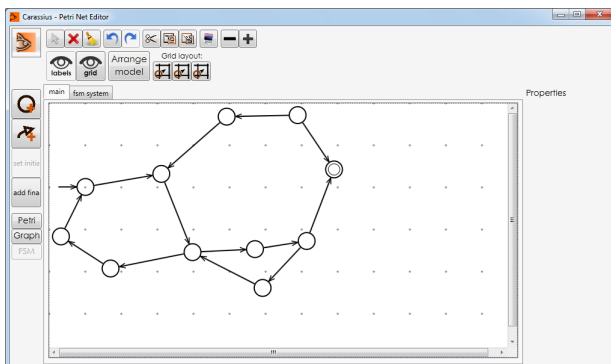


Fig. 3. A finite-state machine editing

Systems of Finite-State Machines: Systems of communicating FSMs are also supported by Carassius. A system of Finite-State Machines may be useful for modelling processes which appear at the same time and have causal dependencies. A Finite-State Machine System deals with some number of FSMs and relations between them. These relations may be of two types: (1) synchronous (two transitions from the FSMs may fire only at the same time) and (2) asynchronous (there is a special state in-between the FMSs called the channel state). Synchronous relations are denoted by simple lines between two models, which hold the information about transitions which are fired simultaneously. Asynchronous - by sequence of arrow, place and another arrow, meaning that some action performed in one fsm may have consequences in another.

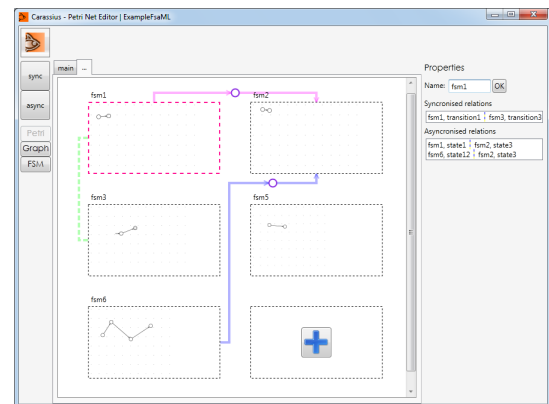


Fig. 4. A system of finite-state machines editing

Import and Export Formats

Carassius provides different import and export formats to facilitate work with models. It deals with several convenient markup language formats for import: PNML for Petri nets, GraphML for graphs, and FSAML for finite-state machines and their systems. All of them are XML-based interchange formats. In addition, one can easily export a model to png-picture or tikz-picture to import model to a \TeX file.

1) **Markup language formats:** PNML and GraphML formats are well-known in the world of modelling and have been in use for a long time. Both of them have a clear specification and will be described further. On the contrary, FSAML (*Finite-State Automaton Markup Language*) has been developed recently by the authors of this paper and has not been formally described yet.

A detailed explanation of a PNML format can be found in [9]. A typical PNML file contains information about a net, a number of pages, lists of places, transitions and arcs. A lot of additional information is available such as names of nodes, dimensions etc. PNML is an extensible format. So, it is possible to make different extensions for particular modelling aspects. It is impossible to cover all extensions. That is why Carassius deals with PNML files according to the recent version of the core standard (ISO/IEC 15909-2:2011).

GraphML is a comprehensive and easy-to-use file format for graphs. It consists of a language core for describing the structural properties of a graph. A detailed description can be found in [10]. Carassius, in turn, supports only simple graphs (directed, undirected and mixed) without any additional features.

FSAML is a format allowing exchange of finite-state machines and their systems. The development of this format is still in progress. However, there is a working alpha-implementation of it in Carassius.

The structure of the file according to the format is following: the main node (`fsasystem`) consists of its name (`name`), a number of finite-state machines (`fsa`), synchronous (`syncs`) and asynchronous (`channels`) relations between them. In turn, a `fsa` node contains a number of states (`state`) and transitions (`transition`). Each of them has an attribute `id` holding unique id. Each state has its type: `general`, `initial` or `final`, therefore there is an inner node `statetype` containing this information. The second inner node is `graphics` representing the data about position and dimension of a node. Transitions have their source states (`source`) and target states (`target`) represented as attributes. The `channels` node consist of several channels (`channel`), which, in turn, have two nodes: `from` and `to` containing information about `fsa` and a corresponding state. The `syncs` node has the same structure except the fact that relation is between two transitions, not states.

An example of the file in the FSAML format is shown in listing 1.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <fsaml>
3   <fsasystem id="fsasystem1">
4     <name>
5       <text>ExampleFsaSystem</text>
6     </name>
7     <fsa id="fsm1">
8       <state id="state1">
9         <statetype>
10          <text>initial</text>
11        </statetype>
12        <graphics>
13          <position x="37" y="68" />
14          <dimension x="30" y="30" />
15        </graphics>
16      </state>
17      <state id="state2">
18        <statetype>
19          <text>general</text>
20        </statetype>
21        <graphics>
22          <position x="107" y="62" />
23          <dimension x="30" y="30" />
24        </graphics>
25      </state>
26      <transition id="transition1"
27        source="state1" target="state2" />
28    </fsa>
29    <fsa id="fsm2">
30      <state id="state3">
31        <statetype>
32          <text>general</text>
33        </statetype>
34        <graphics>
35          <position x="49" y="17" />

```

```

36    <dimension x="30" y="30" />
37  </graphics>
38 </state>
39 <state id="state4">
40   <statetype>
41     <text>general</text>
42   </statetype>
43   <graphics>
44     <position x="97" y="24" />
45     <dimension x="30" y="30" />
46   </graphics>
47 </state>
48 <transition id="transition2"
49   source="state3" target="state4" />
50 </fsa>
51 <channels>
52   <channel id="channel1">
53     <from>
54       <fsa id="fsm1">
55         <state id="state1" />
56       </fsa>
57     </from>
58     <to>
59       <fsa id="fsm2">
60         <state id="state3" />
61       </fsa>
62     </to>
63   </channel>
64 </channels>
65 <syncs>
66   <sync id="sync1">
67     <from>
68       <fsa id="fsm1">
69         <transition id="transition1" />
70       </fsa>
71     </from>
72     <to>
73       <fsa id="fsm2">
74         <transition id="transition2" />
75       </fsa>
76     </to>
77   </sync>
78 </syncs>
79 </fsasystem>
80 </fsaml>

```

Listing 1. FSAML format

2) *T_EX* and *PNG* export: The tool has features for *T_EX* and *PNG* export. Carassius may generate a code to import picture using *tikz*-package into your *T_EX* file. Figure 5 shows a simple Petri net edited with Carassius and exported directly into *T_EX*. This feature has been implemented with help of N. Chuykin (student at HSE).

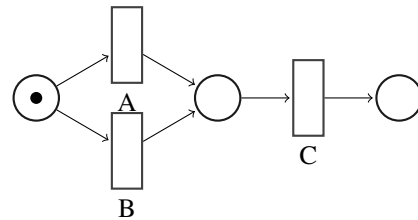


Fig. 5. Picture compiled with *tikz*

In addition, it is possible to export the model as a *PNG* picture.

C. Visualization refinement

The presented tool has several features to make model visualization better. There are two special algorithms for the directed graphs and for Petri nets, which can arrange nodes to make model easier to understand. Graphs and Petri nets can be processed in different ways. The tool also provides a grid for working area which helps placing nodes more accurately. Finally, Carassius provides possibility to hide/show grid as well as node labels. This section describes the layout algorithms in detail.

Algorithm 1: Petri net layout algorithm

Data: List of all nodes as nodes
Result: All nodes are arranged

```

1 int modelNumber = 1;
2 while each node doesn't belong to any model do
3   Node firstNode = findNodeWithoutModelNumber();
4   depthFirstSearch(firstNode, modelNumber);
5   modelNumber++;
6 end
7 foreach model do
8   List<Node> modelNodes =
     getAllModelNodes(modelNumber);
9   List<Node> initialNodes =
     searchForInitialNodes(modelNodes);
10  setColumnForStartingNodes(startingNodes);
11  setColumnForEachNode(modelNodes);
12  setYcoordinateForEachNode(modelNodes);
13  setSpaceBetweenColumns();
14 end
15 visualize();
16 return coordY

```

Petri net layout: Firstly, the layout refinement algorithm for Petri nets is described. It is a layered-based algorithm which was developed especially for Petri nets. Layered-based algorithms is a group of layout algorithms which work with directed graphs and take their hierarchical structure into account [11]. We chose this approach as the most suitable for Petri nets as they are *directed*, and *bipartite*. The structure of the Petri nets notation is quite suitable for a layered representation. The main scheme of the layered-based approach is described in [12]. These algorithms are aimed to cover the list of aesthetic points:

- (1) single edges direction,
- (2) occupied area minimization,
- (3) uniform nodes allocation,
- (4) long edges avoidance,
- (5) edges-crossing minimization.

Although some of these points may conflict with each other, the approach is viable. It works using three steps:

- (1) allocation of nodes on layers in a way which ensures that edges have single direction;
- (2) choice of the nodes order on layers with the aim of edges-crossing minimization;
- (3) determination of node coordinates on layers with the aim of edges-length minimization.

In the presented algorithm these three ideas are used, but some features are added and changed as well.

Algorithm 2: Determination of all the nodes in model

Data: Initial node as node, number of model as modelNum

Result: All nodes of the model are marked

```

1 foreach Arc arc in node.thisArcs do
2   Node next;
3   if arc.To == node then
4     next = arc.To;
5   else
6     next = arc.From;
7   end
8   next.modelNumber = modelNum;
9   next.isChecked = true;
10  foreach Arc arc1 in next.thisArcs do
11    Node next1;
12    if arc1.To == node then
13      next1 = arc1.To;
14    else
15      next1 = arc1.From;
16    end
17    if next1.isChecked == false then
18      next1.isChecked = true;
19      next1.modelNumber = modelNum;
20      depthFirstSearch(next1, modelNum);
21    end
22  end
23 end

```

The algorithm in Carassius takes into account: (1) a biparticity of Petri nets, (2) the fact that they have directed arcs, and (3) a presence of initial places.

Generally, it determines connected components of a model (a number of individual graphs in one model), applies layered-based approach for each component and then gathers components together to visualize an overall model. We use so-called 'columns' to represent layers. Due to the Petri nets biparticity the content of columns alternates from places to transitions. We start from the first column with places. When several steps of the algorithm are made, each node has its column (using breadth-first search), and we can arrange nodes in each column separately (set them y-coordinate). The overall algorithm 1 shows all the steps.

In order to arrange nodes the tool makes the following steps:

(a) Determines connected components of the models. A Petri net model may consist of several individual connected components, so we have to detect them. Also, for each set of nodes we have to assign the number used for component identification.

Next steps are done for each connected component of the model:

(b) Finds all initial nodes (both transitions and places). A node considers as initial if it doesn't have any ingoing arcs.

(c) Sets columns for the initial nodes. This step is needed because these nodes will become starting points to move through the graph.

(d) Sets a column for each node. This algorithm is layered-based, thus, we need to distribute nodes among columns.

(e) Sets a y-coordinate for each node. At this step we want to place each node in some place at a column. To make the

model layout more compact we locate nodes symmetrically from the centre of a column (mean value between minimal and maximal y-coordinate of nodes in a column).

(f) Sets margin between columns. There may be very few or, on the contrary, too many arcs between the nodes in two adjacent columns. So, these distances should depend on a number of arcs between neighbour columns.

(g) Visualizes the whole model. The whole model is visualized using all information derived at the previous steps.

The listing 2 shows the algorithm which divides a model into several connected components. To obtain the list of initial nodes the algorithm 3 is used.

Algorithm 3: Search for the initial nodes

Data: List of all nodes as nodes
Result: List of initial nodes as initialNodes

```

1 List<Node> initialNodes = new List<Node>(); foreach Node
  node in nodes do
2   if node.thisArcs.Count == 0 then
3     initialNodes.Add(node);
4   else
5     bool hasIngoingArcs = false;
6     foreach Arc arc in node.thisArcs do
7       if arc.To == node then
8         hasIngoingArcs = true;
9         break;
10    end
11    end
12    if hasIngoingArcs == false then
13      initialNodes.Add(node);
14    end
15  end
16 end
17 return initialNodes

```

The distribution of all nodes in columns is shown in the algorithm 4.

Algorithm 5 arranges each node for its place (y-coordinate) in a column.

Graph layout: In this subsection the layout algorithm for graphs is described. Carassius contains implementation of the existing algorithm from [13] with little changes. It is a force-directed algorithm aspired to achieve several goals:

- (1) nodes should not be too close to each other,
- (2) edges should have more or less equal length and do not cross each other too often.

This algorithm does a number of iterations to achieve the best arrangement of a graph. It is done by assigning so-called *forces* and *velocities* among the set of edges and the set of nodes, based on their relative positions.

An algorithm for graph layout in Carassius consist of two main steps:

(a) The force-directed algorithm (see algorithm 6) itself. It is applied for each connected component. Constants used in the algorithm were selected experimentally based on application UI configuration.

(b) A movement of all nodes on fixed distances. Nodes can have negative coordinates after applying the algorithm, so we

Algorithm 4: Find a column for each node

Data: List of all nodes as nodes
Result: Each node has its column

```

1 int currentColumn = 1;
2 while each node hasn't its column do
3   List<Node> currentColumnNodes = new List<Node>();
4   foreach Node node in nodes do
5     if node.column == currentColumn then
6       currentColumnNodes.Add(node);
7     end
8   end
9   foreach Arc arc in node.thisArcs do
10    Node temp;
11    if arc.To == node then
12      temp = arc.From;
13    else
14      temp = arc.To;
15    end
16    if node.column == 0 then
17      node.column = currentColumn + 1;
18    end
19  end
20  currentColumn++;
21 end

```

Algorithm 5: Set a position to each nodes in a column

Data: Current column as column, maximum number of elements in column for all model as maxNumberOfElements, list of all nodes in one model as modelNodes
Result: Each node in column has its own y-coordinate

```

1 int numberOfElementsInColumn = 0;
2 foreach Node node in modelNodes do
3   if node.column == column then
4     numberOfElements++;
5   end
6 end
7 double coordY = cellHeight / 2 * (maxNumberOfElements -
  numberOfElements);
8 foreach Node node in column do
9   node.Y = coordY;
10  coordY += cellHeight;
11 end

```

need to move them because working area shows only those which have positive coordinates. We also need to do some movements to place models in such a way in order to save a distance between them.

D. Simulation

Petri nets are not only simple bipartite graphs but also a powerful tool able to represent a process flow. There are 'tokens' (markers inside places), reflecting current state of a net. They can change their places by the transitions firing. A transition may be fired if all places which have outgoing arcs to this transition have enough tokens inside (equal or more than weight of a corresponding arc). At each step only one transition is fired (may be chosen by hand or randomly). When a transition is fired it consumes the required number of tokens

Algorithm 6: Force-based algorithm for a graph model layout

Data: List of all nodes in one model as nodes, list of all arcs in one model as arcs

Result: All nodes in one model are arranged

```

1 double oldX, oldY, newX, newY;
2 foreach Node node in nodes do
3   // nextDouble returns a real number from 0 to 1 node.X =
4   // 200 + nextDouble() * 300;
5   node.X = 200 + nextDouble() * 300;
6   node.Y = 100 + nextDouble() * 200;
7 end
8 do
9   for i ← 0 to nodes.Count do
10    nodes[i].netForceX = nodes[i].netForce.Y = 0;
11    for j ← 0 to nodes.Count do
12      if i == j then
13        continue;
14      end
15      double squaredDistance =
16        (node[i].X - node[j].X)2 +
17        (node[i].Y - node[j].Y)2;
18      nodes[i].netForceX += 200 * (nodes[i].X -
19        nodes[j].X) / squaredDistance;
20      nodes[i].netForceY += 200 * (nodes[i].Y -
21        nodes[j].Y) / squaredDistance;
22    end
23    foreach Arc arc in arcs do
24      Node tempNode;
25      if arc.From == nodes[i] then
26        tempNode = arc.To;
27      else
28        tempNode = arc.From;
29      end
30      nodes[i].netForceX += 0.06 * (tempNode.X -
31        nodes[i].X);
32      nodes[i].netForceY += 0.06 * (tempNode.Y -
33        nodes[i].Y);
34    end
35    nodes[i].velocityX = (nodes[i].velocityX +
36      nodes[i].netForceX) * 0.85;
37    nodes[i].velocityY = (nodes[i].velocityY +
38      nodes[i].netForceY) * 0.85;
39  end
40  oldX = nodes[0].X;
41  oldY = nodes[0].Y;
42  foreach Node node in nodes do
43    node.X += node.velocityX;
44    node.Y += node.velocityY;
45  end
46  newX = nodes[0].X;
47  newY = nodes[0].Y;
48 while oldX != newX || oldY != newY;
  
```

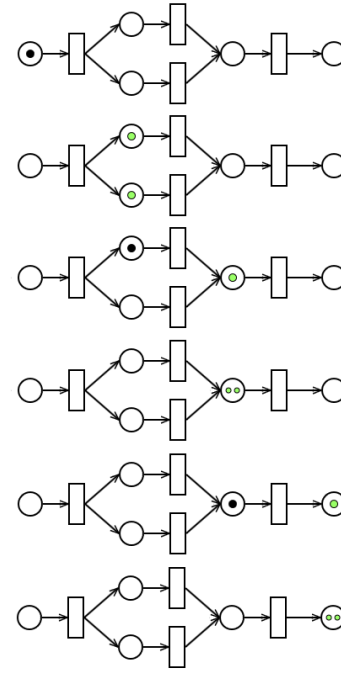


Fig. 6. Simulation of a Petri net

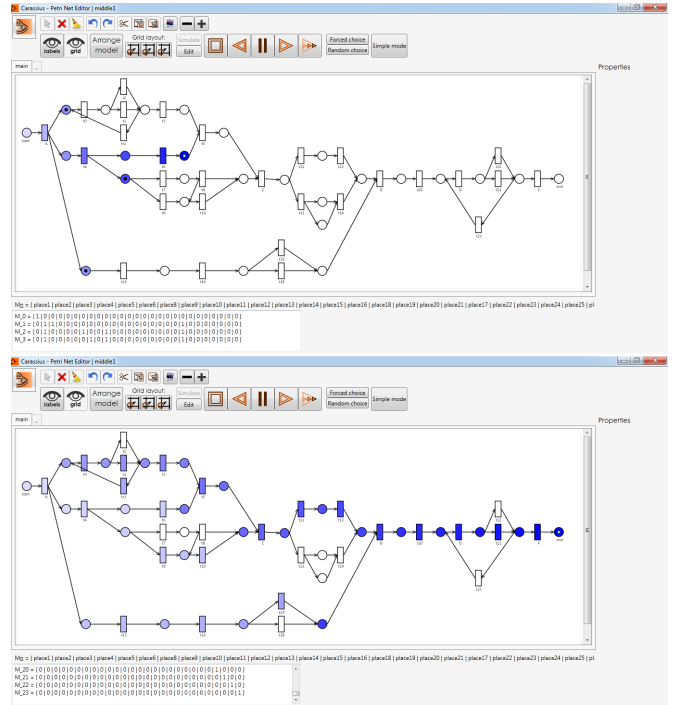


Fig. 7. Wave colouring of a simulation

and passes a token to each outgoing place. The simulation ends when there is no transition able to be fired.

Simulation of an example Petri net made in Carassius is shown in figure 6.

Wave Colouring: Simulation of a net in our tool may also be done in a waving mode. During simulation nodes are coloured in a specific way. A movement of a token from one place to another will be considered as a single step. Nodes engaged in the last step has deep blue colour, whereas

nodes used in previous steps are coloured in light blue. In other words, the later a step is made, the darker a node is coloured, the earlier – the lighter. This colouring allows for easily understanding of a process direction, determining which nodes were visited and which were not.

Figures 7 show how wave colouring of a simulation works in Carassius. The top part of the picture shows simulation at the intermediate step. The bottom part shows a window when the simulation has been ended.

E. Architecture

The tool is built as a standalone windows application using C#. We used the Windows Presentation Foundation (WPF) platform to build our application because of its functionality, extensibility and convenience. The WPF provides user controls as a mechanism for reusing blocks of the UI elements. The main window of Carassius consists only of one user control, which may be easily moved to another application as a component.

III. RELATED WORK

A variety of model editors are available now. Nevertheless, all of them did not fully meet our two main requirements (simplicity and extensibility). This section describes the closest existing tools which support model editing in a desirable way.

a) *CPN Tools* (see [14]): CPN Tools is a tool for working with Colored Petri nets. It allows users to edit, simulate, and analyse them. CPN Tools has an interesting, original interface which uses a lot of small inner windows for each type of editing. However, at first a user can get stuck because the GUI is not very intuitive and the user needs to read the help to understand what he should do in order to start working. In addition, the tool works only with coloured Petri nets and you cannot work with simple ones.

b) *Yasper* (see [15]): Yasper, as authors say, is the yet another smart process editor. It is a quite simple, but useful tool which supports editing and simulation of Petri nets. It has rather user-friendly and easy to use interface, but it is still unevident how to do some actions. Fortunately, its help paper is very useful and provides a lot of information about usage of the tool. However, Yasper has a significant drawback - it does not support the current version of the PNML format, so the user just cannot download new PNML files and cannot work with exported files from the tool anywhere else.

c) *Tina* (see [16]): Tina is a tool for working with classical P/T and Time Petri nets. It has features for editing and analysis of Petri nets. Tina's interface is very simple, but at the same time easy to understand. Editing functionality is not very wide, but the tool provides several analysis techniques, which work well. Tina's disadvantage is that it cannot simulate Petri nets in a visual way and has a small number of functions.

We can see that several tools for working with Petri nets are already exist, but all of them have certain drawbacks. In our tool we endeavoured to take into account all disadvantages we found in other tools, and at the same time to add new functionality. We tried to do interface easy to use and *learnable*, intuitive to work; to provide support of different export and import formats; to implement all main tasks which can be done with Petri nets; and, finally, to incorporate some new features (e.g. several visualisation refinement algorithms).

IV. CONCLUSION

A lot of features and several modes are already implemented in Carassius. One can use it to deal with graphs, Petri nets, Finite-State Machines. Due to modularity of the tool we want to extend it with other modelling formalisms. The most difficult thing is to preserve the simplicity of the software while adding new features.

Our tool has been used in different other projects at PAIS Lab [17], [18]. We hope, it will also be useful for other researchers (see [19]).

Of course, there is still a lot of work to do. Our main goal is to improve the FSM aspect of the tool. This functionality is involved in other projects of our group. Complete definition of the FSAML format is the key point of the future work. Moreover, we intend to add a simulation functionality for the finite-state machines.

Another aim is to carry out a number of user tests in order to find and eliminate bugs in the tool. In addition, we are going to do usability testing to make Carassius more intuitive to use and work with. There are several possible improvements of GUI we want to implement.

ACKNOWLEDGMENT

We would like to thank members of the PAIS Lab for their support. Research assistants I. Shugurov and A. Begicheva tested the tool and reported lots of bugs. Dr. A. A. Kalenkova and prof. I. A. Lomazova gave us a valuable advice on the GUI design and the required features.

Also we would like to thank Nikolay Chuikin, who implemented the \TeX -export used in the tool.

This work is output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE).

REFERENCES

- [1] M. Dumas, W. M. van der Aalst, and A. H. ter Hofstede, *Process-aware Information Systems: Bridging People and Software Through Process Technology*. New York, NY, USA: John Wiley & Sons, Inc., 2005.
- [2] M. Weber and E. Kindler, "The petri net markup language," in *Petri Net Technology for Communication-Based Systems - Advances in Petri Nets*, 2003, pp. 124–144.
- [3] J. Billington, S. Christensen, K. M. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber, "The petri net markup language: Concepts, technology, and tools," in *Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003, Eindhoven, The Netherlands, June 23-27, 2003, Proceedings*, 2003, pp. 483–505.
- [4] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. Marshall, "Graphml progress report structural layer proposal," in *Graph Drawing*, ser. Lecture Notes in Computer Science, P. Mutzel, M. Jnger, and S. Leipert, Eds. Springer Berlin Heidelberg, 2002, vol. 2265, pp. 501–512.
- [5] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall, "Graphml progress report," in *Graph Drawing*, 2001, pp. 501–512.
- [6] W. Reisig, *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.
- [7] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
- [8] J. A. Anderson, *Automata theory with modern applications*. Cambridge University Press, 2006.
- [9] L. Hillah, E. Kindler, F. Kordon, L. Petrucci, and N. Treves, "A primer on the petri net markup language and iso/iec 15909-2," *Petri Net Newsletter*, vol. 76, pp. 9–28, 2009.

- [10] U. Brandes, M. Eiglsperger, and J. Lerner, "Graphml primer," *Online: <http://graphml.graphdrawing.org/primer/graphml-primer.html>* [29.05.2007], 2004.
- [11] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.
- [12] V. Kasianov and V. Evstigneev, *Grafi v programirovanii*. BHV - Peterburg, 2003.
- [13] S. G. Kobourov, "Spring embedders and force directed graph drawing algorithms," *arXiv preprint arXiv:1201.3011*, 2012.
- [14] M. Westergaard and L. M. Kristensen, "The access/cpn framework: A tool for interacting with the cpn tools simulator," in *Applications and Theory of Petri Nets*. Springer, 2009, pp. 313–322.
- [15] K. van Hee, O. Oanea, R. Post, L. Somers, and J. M. van der Werf, "Yasper: a tool for workflow modeling and analysis," in *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*. IEEE, 2006, pp. 279–282.
- [16] B. Berthomieu*, P.-O. Ribet, and F. Vernadat, "The tool tina—construction of abstract state spaces for petri nets and time petri nets," *International Journal of Production Research*, vol. 42, no. 14, pp. 2741–2756, 2004.
- [17] A. K. Begicheva and I. A. Lomazova, "Checking conformance of high-level business process models to event logs," in *Proceedings of the Spring/Summer Young Researchers Colloquium on Software Engineering*, vol. 8, 2014.
- [18] A. A. Mitsyuk and I. S. Shugurov, "On process model synthesis based on event logs with noise," *Modeling and analysis of information systems*, vol. 4(21), pp. 181–198, 2014.
- [19] N. Nikitina, "Carassius: A Simple Petri Net Editor," accessed: 2015-04-01. [Online]. Available: www.pais.hse.ru/research/projects/carassius

Iskra: A Tool for Process Model Repair

Ivan S. Shugurov, Alexey A. Mitsyuk

Laboratory of Process-Aware Information Systems

National Research University Higher School of Economics

3 Kochnovsky Proezd, Moscow, Russia

Email: shugurov94@gmail.com, amitsyuk@hse.ru

Abstract—This paper is dedicated to a tool which is aimed to facilitate process mining experiments and evaluation of the repair algorithms. Process mining is a set of approaches which provides solutions and algorithms for discovery and analysis of business process models based on event logs. Process mining has three main areas of interest: model discovery, conformance checking and enhancement. The paper focuses on the latter. The goal of enhancement process is to refine an existing process model in order to make it adhere event logs. The particular approach of enhancement considered in the paper is called decomposed model repair. Although the paper is devoted to the implementation part of the approach, theoretical preliminaries essential for domain understanding are provided. Moreover, a typical use case of the tool is shown as well as guides to extending the tool and enriching it with extra algorithms and functionality. Finally, other solutions which can be used for implementation of repair schemes are considered, pros and cons of using them are mentioned.

Index Terms—Process model, Petri net, Model repair, Process mining.

I. INTRODUCTION

In this paper, a tool for modular process model repair is presented. Architectural features and usage examples are provided.

Process mining [1] is a research area which deals with analysis of information systems or business processes by studying corresponding event logs and building process models. The basic idea is that there can be significant improvements of existing systems, business operations if event logs and their content are studied more thoroughly. Three main aims of process mining are *process discovery*, *conformance checking* and *enhancement* [2].

The goal of *Process discovery* is to create a process model, based on an event log. That constructed model has to adequately describe the behavior observed in the event logs. The process of construction is typically called mining. As a model it is possible to use, for example, Petri nets. The challenge of process discovery is the hard truth that event logs reflect only a fraction of the overall process. It means that there may be activities, events, conditions, decision forks which exist in the initial process model, but they are not seen in event logs. For example, rare events in processes such as activities undertaken in emergency situations at nuclear power stations. Such activities exist, they are strictly regulated by rules and legislation, they influence the overall process a lot, but they are extremely uncommon so if an event log of a nuclear station is considered they are likely not to be present. Another serious issue concerning event logs is errors in them. Some events may

be not put down in logs, log records might contain incorrect information about actually occurred events (i.e. wrong time stamp, event name), they might be deliberately distorted.

Conformance checking is aimed to check whether a model fits a given event log. Because of the reasons presented in the description of process discovery, perfect fitness is almost not feasible in practice. Therefore, when discrepancy between a model and corresponding event logs occurs, it is desired to assess the significance of the deviation and highlight model parts where deviations take place [3], [4]. Some types of conformance checking algorithms support assigning weights to skipping and adding of events, that somehow indicates the significance of these deviations.

The reason for applying *Enhancement* is to improve already existing models by using information stored in event logs. So, the task here is to alter model, not to create an absolutely new one. Typical input parameters for the enhancement algorithms are a model and a corresponding event log. According to the presented definition, the approach the tool implements is categorized as an enhancement approach.

The remainder of this work is organized as follows. In section II basic ideas behind model repair are described. III sections explains what modular repair is and how tools implementing this approach should be organized in order to achieve the goals. In section IV a summary of the tool functionality is reported. Section V contains information on the framework used during the development process, domain analysis and the architecture of the tool. Section VI shows step-by-step usage of the tool. In section VII other tools are considered. Section VIII concludes the paper.

II. PROCESS MODEL REPAIR

In the field of process modeling not all the processes are of best quality. Usually process models are made by experts or obtained as a result of using automated model construction algorithms. In the field of process mining a lot of methods have been developed to discover models from process logs[1]. Real-life processes in information systems are complex and permanently changing. Thus, it is a very hard problem for experts and engineers to keep process models up to date.

The goal of model repair is to improve the quality of a model. In this paper, fitness is understood as a metric of model quality. Fitness is measured using technique described in [3]. Model repair has been introduced in [5]. As input for model repair a process model M and an event log L are used. If

model M conforms to L , then there is no need to change M . If M partially does not conform to L , repair method has to repair non-conforming parts of M . Conforming parts of the model are kept as is. The result is the repaired model M' .

III. MODULAR REPAIR APPROACH

The implementation of the modular repair approach is the foundational goal of this work. The key idea is to make a general model repair scheme which will consist of several cells connected with strong links. A *cell* is understood as a placeholder where a user can put one of the appropriate algorithms. Cells are of the following types: (1) conformance checking cell, (2) decomposition cell, (3) selection cell, (4) repair cell, (5) composition cell, (6) final evaluation cell. Each cell type corresponds to the step in the modular repair. The schema of cells is provided in figure ??.

Conformance checking cell is used to evaluate current progress of the repair process and indicate whether a current model quality is sufficient. An algorithm in a *decomposition cell*, as it is clear from its name, is responsible for dividing an entire model into smaller parts, which are easier to understand, analyze and repair. Decomposition for process mining is described in [6]. A *selection cell* includes an algorithm whose aim is to choose run conformance checking for each model part and decide which of them are sufficiently fit. A *repair cell* can be either a process discovery algorithm or some enhancement algorithm, although for generalization reasons they are called repairers in the paper. Once the decomposed parts are repaired they ought to be merged in order to form a single model. It is done by an algorithm located in a *composition cell*. An algorithm located in a *final evaluation cell* is executed after completion of the entire repair task. At this step several metrics are measured in order to assess the quality of the model and the repair. Moreover, similarity of the initial and the final models is checked. In the future, visualization of model differences will be incorporated.

At the first step the tool checks whether a model and a log conform to each other. The second step is one of the model decomposition methods, which allows for splitting the model into parts [7]. At the third step the tool selects conforming and non-conforming parts by application of conformance checking method to each part, obtained at the second step with the projection of the event log onto set of activities corresponding to this part. The fourth step is the repair step. At this step tool applies the repair algorithm. It can be, for example, simple re-discovery algorithm. By applying it the tool obtains a new model from the log part corresponding to a non-conforming part of the initial model. At the fifth step the tool composes all parts of the model into the final model using an appropriate method. The sixth step is the final evaluation of the repaired model. Usually, each algorithm has to be wrapped in additional code in order to be embedded in a particular cell of the tool.

This work will not consider the aspects of the methods which can be placed into cells. There is a theory behind each step of the repair process. Methods offer a lot of settings and options. So, it will be impossible to put all the details in one

text. The main goal of this work is to propose a software architecture that allows for exploring different algorithms and their features in the context of model repair.

IV. TOOL OVERVIEW

The main functionality provided by the tool implies the following aspects:

- The tool allows users to select a decomposition method which, in their opinion, is the most suitable for a given model.
- The tool makes it possible to choose a repair algorithm. The choice of the algorithm is typically based on the properties of the algorithm and a model it produces. The task of choosing the best repair algorithm is basically an attempt to find appropriate alternative between time needed for the algorithm to do its work, presence or absence of so-called silent transitions (i.e. transitions that do not correspond to any events observed in an event log, but considered to be present because they somehow explain the model behavior) and conformance between a given model and an event log.
- One may specify importance of each metric for a particular repair task. This step is essential for automatic evaluation of how well the tool helps researchers achieve the desired repair result.
- Numeric results of the final model evaluation can be stored in CSV file either manually or automatically. CSV files are chosen because a lot of tools support this format, therefore, it significantly simplifies the further analysis or visualization. The evaluation process assesses the following metrics: fitness (two approaches for fitness measurement are employed), conformance, complexity and a similarity coefficient.
- The tool is responsible for visualization of each step the tool performs and a final model. In the future, the tool will also be fitted with a convenient visualization of the difference between an initial and a final model.
- The tool makes it possible to significantly modify logic the cells use, thus extending the tool or adjusting it to a particular circumstance.

It goes without saying that despite the existence of some theoretical guidelines, choosing the right decomposition and repairing algorithms as well as their settings can be extremely complicated and mean, in the worst-case scenario, brute-force seeking the right methods. Because of that, one of the tool's aim is to facilitate this very tedious process. Moreover, if one is developing or evaluating a repair algorithm, it will imply a lot of repetitive executions of it. Hence, the tool facilitates this process a lot and is likely to significantly reduce time spent on such tasks.

V. TOOL ARCHITECTURE

A. ProM

The tool is being developed using Java 6 Standard Edition and ProM 6.4 Framework [8]. ProM 6.4 is an open-source

framework specially designated for implementing process mining algorithms in a standardized way. ProM 6.4 consists of the core part and disjoints parts called plugins. The core part of the framework is responsible only for uploading available plugins, handling plugins life cycle, interaction between plugins and basic functions for dealing with graphical user interface. Hence, programmers focus exclusively on implementation of algorithms, work with file system and visualization of results. The framework is distributed under GNU Public License, although some plugins are distributed under different licenses.

Once a plugin is properly configured, ProM automatically discovers, finds and uploads it, then this plugin is added to the list of all available plugins. In addition, the list of plugins demonstrates parameters required by each plugin. By doing this, the framework simplifies providing parameters needed for plugins. Nowadays, almost all data types for working with Petri nets have been implemented and supplied with visualizers, so researchers and developers are eliminated of necessity to implement them from scratch.

Each plugin has so-called context. Context acts as a bridge between plugin and the framework because it is the only way plugins can interact and collaborate with ProM. For every context child contexts may be created, each of which is designated for a specific task. Thus, it is possible to construct a hierarchy of plugin calls from a parent plugin.

Plugins may run either with or without graphical user interface. The former provides a rich possibility to interact with user or visualize data, whereas the later enables to call other plugins in the background simultaneously with user interaction in the main plugin. ProM encourages developers to write an extendable and loose coupled software, providing a rich set of tools. One of such tools, extensively used in the tool, is a mechanism for finding all classes annotated in a special way. Arguably the most common way to use annotations is to mark Java classes that contain algorithms. One creates an interface for a set of related algorithms, then annotate each of them. After that, they can be easily found and used via annotations.

Interaction between plugins is accomplished by using a plugin manager. The plugin manager provides API for invoking plugins, makes sure that correct context is configured for a called plugin. The plugin manager enables not only to invoke known plugins but also to look for plugins with specific signature, to invoke them and to obtain results of their executions. Despite its promising flexibility and convenience, in practice it is generally easier to use conventional method invocations, because the API exposed by the plugin manager is a bit unintuitive. Furthermore, direct methods calls ensure more readable code. Because of these reasons, the direct methods call are preferred in the tool and used wherever possible.

The core part of a typical ProM processing plugin is a class which contains at least one public method. This method must have a special annotation which registers it in the ProM framework as a plugin. The name, input and output parameter lists are listed inside the annotation. Particular plugin context

of a current ProM session have to be among the other parameters of the method.

The tool, which implements the approach presented in this work, is built as a plugin for the ProM Framework; therefore architecture of the tool has to fulfill all the aforementioned requirements for ProM plugins. We decided to use such an approach because the framework already has plugins which take care of discovery of Petri nets, event logs import and export, conformance checking as well as decomposition plugins, and provides further opportunities to work with the resulting data.

B. Preliminary domain analysis

This section is completely devoted to the analysis of the existing plugins for decomposition and model repair, because their usage involved extensive and from time to time tricky interaction with ProM and ProM plugin manager. In addition, the way how decomposition and repair model plugins are used is of high importance because it influences whether the tool is easy to extend. Detailed explanation of how conformance checking, final evaluation and the overall infrastructure are made is left to the following subsection.

The core implementation task of this project was to incorporate a dozen of available plugins for model repairing, decomposition and conformance checking, that have different authors, coding styles and settings. One of the main requirements for the resulting architecture was to make it as straightforward and comprehensive as possible, though ensure that it is flexible. In addition, we wanted to reuse as much of the existing code as possible. It meant that before the development of the tool could be started there was a need to scrutinize source code files of existing projects which we intended to use. This analysis was focused on 3 most important questions: (1) Does the architecture of each plugin follow MVC pattern [9]? (2) How heavily does each plugin use ProM-specific classes, tools? For example, can it be easily retrieved from ProM and used as a some sort of standalone application? Do any of plugin show graphical user interface? (3) What set of parameters is required for each plugin?

The conducted analysis of repair algorithms revealed that the source code had been written in inconsistent way, the majority of plugins do not follow the MVC principles, that increased efforts needed for using them. As a result, plugins we intended to use were separated into 3 groups according to their coupling with ProM and the simplicity of their reuse:

- Plugins whose execution needs requesting via the plugin manager of ProM. Hence, in order to call them we supply plugin name, a list of required parameters and types of expected output. Then the plugin manager seeks the requested plugin and executes it. Examples of such plugins are *Alpha miner* [10] and *ILP Miner* [11].
- Plugins whose execution can be initiated via usual Java method calls without need to delegate this task to the ProM plugin manager. *Genetic miner* [12] and *Heuristics miner* [13] are placed in this group of plugins.

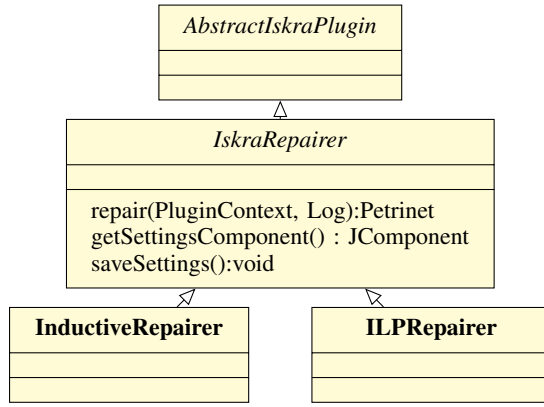


Fig. 1. Repairers hierarchy

- Plugins whose architecture follows the MVC pattern. They are characterized by clear separation of actual algorithm and ProM-specific parts. Such plugins are more desirable because their usage and extension requires less time and effort. Unfortunately, *Inductive miner* [14] is the only plugin which falls into this category.

The subsequent step was to determine the ways which would allow users of the tool to specify parameters for repair algorithms if users wish to do it, otherwise default parameters would have to be set. The study of the plugins showed that only *Alpha miner* does not show GUI, whereas others do but have only one screen with settings, which allows for significant simplification of the resulting design decisions.

The situation with decomposition algorithms is a bit more easier despite some nuances. First of all, they are highly-sensitive to the input data. Event logs may include a lot of information in order to simplify further log analysis and error detection. ProM plugins responsible for projection a net on a log are aware of this information and try to make full use of it while projecting a net. By *projection* in this paper the process of extracting events which correspond to a particular model part from the entire event log is understood. Despite its high purpose, it is prone to produce rather unexpected outcome. It seems they work better and give correct result if event logs contain information only about event names. Concerning this issue it is absolutely essential to apply some kind of model and event log preprocessing techniques before trying to decompose and project a model. Furthermore, model decomposition is typically not a one-step process - it requires a number of consequent plugin calls, but for the sake of simplicity, covering up this circumstance from the main logic of our tool was on the list of the goals.

On the other hand, all decomposer plugins may be executed without showing GUI. In fact, only SESE decomposer [15] has one. Nevertheless, the possibility of existence of GUI was considered thoroughly due to extendability and flexibility matters.

C. Usage of decomposition and repair algorithms

Judging by the results of the analysis of repair plugins we came up with a detailed plan on how to abstract from specific implementation details and provide a common interface for using these plugins. Of course, each of 3 plugin types (model repairing, model decomposition and conformance checking) has its own interface, unique for its specific nature. So, the final decision was to write "wrapper" interfaces and classes for required plugins. *Wrapper* is understood as a class which defines a common interface and hides the details how actual plugin is invoked. In fact, the concept of the adapter pattern [16] was exploited. The tool works only with such wrappers without knowledge how inter-plugin communication is carried out. Furthermore, wrappers apply an idea of using annotations, which allows for complete deliverance from dependencies of the tool on wrappers and, hence, on external plugins. This approach also facilitates extension of the tool: those who are willing to incorporate new algorithms do not need gaining access to the source code of the tool. The only thing that has to be done is to create a Java class that extends either *IskraDecomposer* or *IskraRepairer* and marked by the corresponding annotation (either *@IskraDecomposer* or *@IskraRepairer*). Then ProM will detect this class and our tool will add it to the list of available algorithms. One important constraint is that wrappers must have an empty constructor. If a wrapper does not have it, the wrapper will not be available.

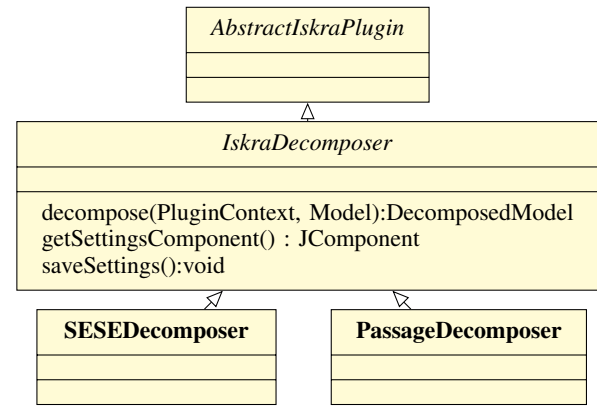


Fig. 2. Decomposers hierarchy

Figure 1 and figure 2 depict the design of repairers and decomposers. Class *AbstractIskraPlugin* is a common super-class for all implemented wrappers. It encapsulates plugin's name and indicates that it is a plugin after all. Then, there are two abstract classes *IskraRepairer* and *IskraDecomposer* which provide a common interfaces respectively for repairers and decomposers. The tool uses only links to these classes, not to their subclasses. The architecture has been implemented and proved to be viable. *InductiveRepairer*, *ILPRepairer*, *SESEDecomposer*, *PassageDecomposer* [17] are examples of actual (not abstract) classes. In order to save space and make a picture more comprehensible only these classes are shown, however half a dozen of others adhere to the architecture and available in the tool.

The typical scenario of using wrappers is:

- 1) method *getSettingsComponent* is invoked.
- 2) if the value returned after the invocation *getSettingsComponent* is not *null*, then received GUI is displayed to a user
- 3) GUI demonstration means having to save setting by invoking *saveSettings* method
- 4) at this point a plugin is properly configured and is ready to be used. Only one thing left to get result - to invoke either (repair) or *decompose*.

It must be mentioned that steps 1-3 are arbitrary. If a user is either satisfied with default setting or does not want to show GUI then according to the contract, a wrapper supplies defaults settings to a corresponding plugin. If a plugin does not have any graphical elements for settings, then *getSettingsComponent* returns *null* and steps 2-3 are skipped. In case of repair algorithms an object of type *DecomposedModel*, which holds parts of the initial model and an event log for each of the parts, is returned.

D. Tool infrastructure

A number of algorithms for conformance checking is really limited in ProM. There are only 2 prominent algorithms: conformance by replay and conformance using alignments, others are mainly variations of mentioned. Thus, there is no urgent need to provide really flexible solution. Both of these algorithms are used in the tool. The algorithm described in [3] is used as a main conformance algorithm in the tool, it is placed in *Conformance checking cell*. In order to allow convenient and user-friendly usage of this algorithm, the corresponding plugin has been changed slightly. The plugin was partly separated from ProM in order to ensure its robustness. Moreover, parameters of the plugin include information on a model which is about to be used and the original parameter creation mechanism does not permit to create it silently, without showing GUI. Because of that reason, parameter classes were supplemented with "copy constructors" which take a new model and copy an existing parameter adhering it to the new model. Another algorithm is provided as an optional add-on and used in a *final evaluation cell*. The usage of this plugin required to slightly change classes related to user interface.

All discussed cells are parts of the abstraction called *repair chain*. A repair chain represent the very nature of the decomposed repair approach. Each chain implies algorithms which correspond to the cell types, then it makes plugin calls in the specified order ensuring the work of the tool. The goal of designing repair chains was to make a good level of abstractions from which algorithms (cells) are used, how they are used, in which order; and to execute every chain with different models without need to reconstruct the chain. In order to achieve these objectives, the idea of dependency injections is heavily exploited. Decomposition and repair plugins are supplied via constructor injection, whereas a model, a conformance checking algorithm and its parameters are provided as a method parameters. This discrepancy has rather ordinary

explanation. Decomposition and repair algorithms represent something stable which can be reused over and over again with different models in a handy manner. In contrast, a model, a conformance checking algorithm and conformance checking parameters are volatile and tightly coupled.

Introducing a new data type which encapsulates cells tend to make the tool more flexible and easier to modify, maintain and extend because of the following reasons.

Using abstract data types and dependency injection during the development ensured that each particular chain may be implemented in a way which differs a lot from others. For instance, repair chains may use different triggers to decide when a repaired model is good enough, although the main reason for having separate repair chains is a fact that there are a few of possible strategies of how to choose a model part to be repaired. Some strategies are straightforward - just take a part with the worst fitness, whereas others may use sophisticated techniques, preprocessing and more intelligent choice. However, details, ins and outs of these strategies are out of scope of the paper due to their theoretical nature, the main point here is to establish that different repair chains are possible and that the tool has to provide capability of introducing new repair chains.

It allows users to create several chains which differ in algorithms used in cells and then run all of them at a time. The feature makes testing of several algorithms and their parameters against the same model a lot faster. In order to achieve it 2 plugins are available. One of them, *Iskra chain generator* is responsible for creating repair chains - one selects desired repair chains, algorithms and their parameters. In contrast with a main plugin which creates a chain and then immediately executes it, chain generator returns a list of configured chains to the ProM resource pool rather than execute them. At the moment when all desired chains are built, one may supply them to *Iskra chain runner* plugin. This plugin takes an arbitrary number of repair chains, a model and a corresponding event log, after that the plugin configures settings of conformance checking and sequentially executes each chain. This functionality has already been implemented, although it needs some refinement and improvements.

In order not to have hard-coded chains and plugins around chains a mechanism of annotations and reflective calls was introduced, as used for decomposition and repair wrappers. It enriches the tool with the ability to load repair chains dynamically. Moreover, it lets other developers and researcher to develop new chains, incorporate them in the tool. A Java class which implements repair chain logic has to extend *RepairChain* interface and be annotated with *@Chain*.

VI. USE CASE

As an example of a usage a simplified version of an iteration of a typical agile development process is considered. All activities of the developers are recorded in event log, thus allowing for keeping track of what the team does and analysis of the development process. Initial business process involves writing and running tests after writing code is completed.

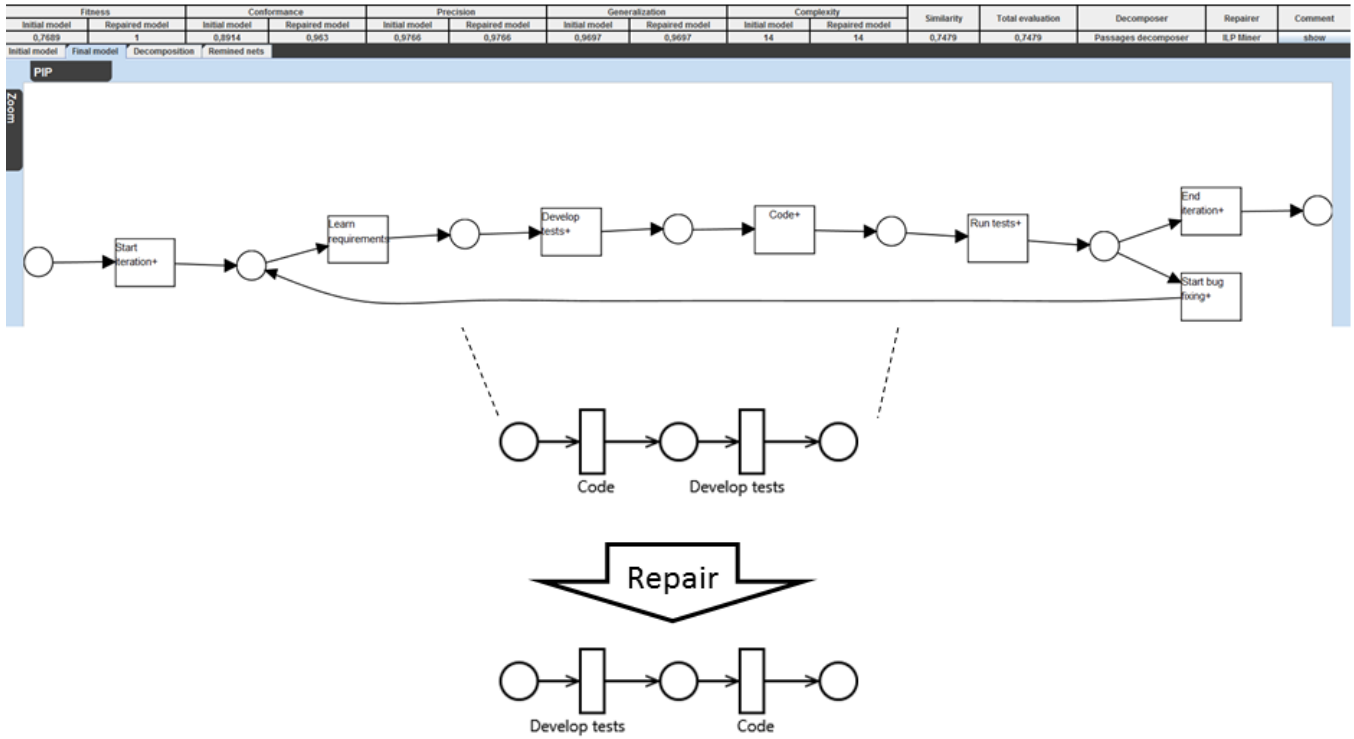


Fig. 3. Illustration of repair

Then, a developers team informally decides to try test-driven development [18], thus creating tests before writing code. These changes are reflected in event logs. After a while a conformance checking algorithm is applied and it reveals that the actual process does not conform to what a company considers as an actual process. Hence, it is necessary to apply repair algorithm in order to learn what has changed and build a proper model of the process.

next step is to select an appropriate repair chain from the list of chains. Afterwards, one is asked to specify setting of each selected algorithm, and after that repair process is executed. Once it is finished, a screen with results is shown, it looks like in figure 3. This screen-shot demonstrates the result of repair and final evaluation of the considered example of agile iteration and clarifies where the change took place and what exactly has changed. As a modeling language Petri nets are applied. It is clear from the screen-shot that fitness increased from 0.7689 to 1, which means that the repair model perfectly fits the given event log and the goal of achieving fitness not smaller than 0.98 has been successfully accomplished. Furthermore, values of others metrics are shown on this screen.

Repairer and decomposer selection			
Decomposition plugins		Repair plugins	
SESE Decomposer	not selected	Alpha miner	not selected
Passages decomposer	selected	Genetic miner	not selected
Maximal decomposer	not selected	Heuristic miner	not selected
Recomposer	not selected	ILP Miner	selected
Stub decomposer	not selected	Inductive miner	not selected
Fitness fitnessThreshold		0.98	

Select one repair chain	
Greedy chain	selected
Naive chain	not selected
Advanced chain	not selected

Fig. 4. Plugin settings

In order to repair a model one needs to select appropriate plugin and supply an existing model and an event log. Plugin's graphical interface used for specifying settings is shown in figure 4. Then one selects a desired algorithms of decomposition and repair. Moreover, one sets minimal fitness a repaired model should have. In the example, desired fitness is 0.98. The

VII. RELATED WORK

The idea of providing a way to chain executions of several plugins or algorithms in a handy way, which is explored in this paper, is also similar to scientific workflow systems. Two of such systems capable of dealing with process mining are considered here.

First tool is *RapidProM* [19] which is a ProM extension for *RapidMiner* [20]. It allows users to build plugin chains in a visual way. Quite a number of ProM Plugins are available in this extension, however not all of them. It can easily be installed via RapidMiner Marketplace. The only question is its ability to be extended. *RapidProM* does not support native ProM plugins and ProM mechanism for loading plugins, therefore plugins come only from the authors of *RapidProM*, which makes the objective of creation and execution of schemes, such

as those discussed in the paper and possible in the presented tool, much harder.

Then comes *DPMine Workflow Language* [21] and *DP-Mine/P framework* which provide a new modeling language which natively supports notion of execution. Implementation of the ideas defined in the language are written in C++ with usage of Qt library. Process models can be constructed using convenient graphical user interface. Furthermore, the solution is intended to be easily extended by adding plugins. The advantage of using C++ is possibility to utilize resources in more effective and flexible way and provide better performance, which is of high importance in the era of Big Data, but the downside is that it cannot be integrated with ProM, so it is deprived of algorithms the ProM system offers.

VIII. CONCLUSION

In this paper, a tool for decomposed model repair is described. Decomposed model repair is used as a way of model enhancement. The tool is implemented as several plugins for the ProM Framework, which guarantees that the tool can be easily distributed and used by both researchers and developers within ProM community. The way the tool is written allows for fast improvement and enhancement of it.

While developing the tool advantages and disadvantages of existing tools were examined. The tool does not have some drawbacks typical for its counterparts. However, there is still room for improvements. In the future the tool will be fitted with more sophisticated mechanism of repair chains. Furthermore, a handy visualization of differences between initial and repaired models, some kind of recommender systems which suggests better repair options according to properties of a model and an event log will possibly be developed and incorporated.

ACKNOWLEDGMENT

This work is output of a research project implemented as a part of the Basic Research Program at the National Research University Higher School of Economics (HSE). Authors would like to thank all the colleagues from the PAIS Lab whose advice was very helpful in the preparation of this work.

REFERENCES

- [1] Wil M. P. van der Aalst, *Process mining: discovery, conformance and enhancement of business processes*. Springer, 2011.
- [2] IEEE Task Force on Process Mining, "Process mining manifesto," in *Business Process Management Workshops*, ser. Lecture Notes in Business Information Processing, F. Daniel, K. Barkaoui, and S. Dustdar, Eds., vol. 99. Springer-Verlag, Berlin, 2012, pp. 169–194.
- [3] W. M. P. van der Aalst, A. Adriansyah, and B. F. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisc. Rev.: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.
- [4] A. Rozinat and W. M. van der Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.
- [5] D. Fahland and W. van der Aalst, "Repairing process models to reflect reality," in *Business Process Management*, ser. Lecture Notes in Computer Science, A. Barros, A. Gal, and E. Kindler, Eds. Springer Berlin Heidelberg, 2012, vol. 7481, pp. 229–245.

- [6] W. M. P. van der Aalst, "Decomposing petri nets for process mining: A generic approach," *Distributed and Parallel Databases*, vol. 31, no. 4, pp. 471–507, 2013.
- [7] W. M. Van Der Aalst, "A general divide and conquer approach for process mining," in *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on*. IEEE, 2013, pp. 1–10.
- [8] Prom framework. [Online]. Available: <http://www.promtools.org/doku.php>
- [9] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented Software Architecture: A System of Patterns*. New York, NY, USA: John Wiley & Sons, Inc., 1996.
- [10] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.
- [11] J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Applications and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, K. van Hee and R. Valk, Eds. Springer Berlin Heidelberg, 2008, vol. 5062, pp. 368–387.
- [12] W. van der Aalst, A. de Medeiros, and A. Weijters, "Genetic process mining," in *Applications and Theory of Petri Nets 2005*, ser. Lecture Notes in Computer Science, G. Ciardo and P. Darondeau, Eds. Springer Berlin Heidelberg, 2005, vol. 3536, pp. 48–69.
- [13] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, "Process mining with the heuristics miner-algorithm," *Technische Universiteit Eindhoven, Tech. Rep. WP*, vol. 166, pp. 1–34, 2006.
- [14] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from incomplete event logs," in *Application and Theory of Petri Nets and Concurrency*, ser. Lecture Notes in Computer Science, G. Ciardo and E. Kindler, Eds. Springer International Publishing, 2014, vol. 8489, pp. 91–110.
- [15] J. Munoz-Gama, J. Carmona, and W. M. van der Aalst, "Single-entry single-exit decomposed conformance checking," *Information Systems*, vol. 46, pp. 102 – 122, 2014.
- [16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.
- [17] W. van der Aalst, "Decomposing process mining problems using passages," in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, S. Haddad and L. Pomello, Eds. Springer Berlin Heidelberg, 2012, vol. 7347, pp. 72–91.
- [18] Beck, *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [19] Rapidprom. [Online]. Available: <http://www.rapidprom.org/>
- [20] Rapidminer. [Online]. Available: <https://rapidminer.com/>
- [21] S. Sergey, "Dpmine/c: C++ library and graphical frontend for dpmine workflow language," *Proceedings of the Spring/Summer Young Researchers Colloquium on Software Engineering*, vol. 8, 2014.

Comparing process models in the BPMN 2.0 XML format

Sergey Y. Ivanov
School of Software Engineering
NRU – Higher School of Economics
Moscow, Russian Federation
syuiivanov@gmail.com

Anna A. Kalenkova
School of Software Engineering
NRU – Higher School of Economics
Moscow, Russian Federation
akalenkova@hse.ru

Abstract - Comparing business process models is one of the most significant challenges for business and systems analysts. The complexity of the problem is explained by the fact there is a lack of tools that can be used for comparing business process models. Also there is no universally accepted standard for modeling them. EPC, YAWL, BPEL, XPD and BPMN are only a small fraction of available notations that have found acceptance among developers. Every process modeling standard has its advantages and disadvantages, but almost all of them comprise an XML schema, which defines process serialization rules. Due to the fact that XML naturally represents hierarchical and reference structure of business process models, these models can be compared using their XML representations. In this paper we propose a generic comparison approach, which is applicable to XML representations of business process models. Using this approach we have developed a tool, which currently supports BPMN 2.0 [1] (one of the most popular business process modeling notations), but can be extended to support other business process modeling standards. This paper is an ongoing research conducted in the frame of a bachelor diploma in the software engineering field.

Keywords: *business process modeling, business process comparison, BPMN 2.0 (Business Process Model and Notation), XML (eXtensible Markup Language), process mining.*

I. INTRODUCTION

The availability of methods and tools capable to compare process models is crucial for business process analysts. Thus, for example, there can be a need to use comparing methods in order to find duplicates in repositories of process models. Finding duplicates is an essential task for those process analysts who wish to add a new process model to a process repository or even merge two repositories. The other obvious example is a comparison of a real and a reference process models. A challenge here is to obtain a real process model. This problem can be solved in several ways, but the most effective known approach is a process model discovery. A new scientific discipline, process mining, can be applied for this purpose. The first type of process mining techniques, discovery, is used to construct models from event logs created by information systems [2].

Since the process model is discovered, we have a reference and a real process models. After that, we can move to the comparison of these two process models (Fig. 1).

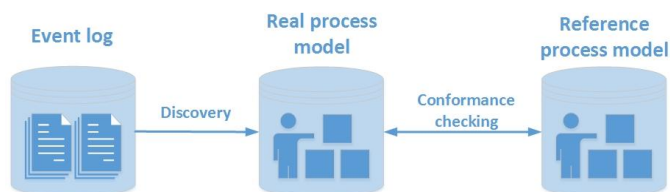


Figure 1. Conformance checking between two process models

The following approaches for comparing business process models are currently known: lexical matching, structural matching, and behavioral matching.

Lexical matching is based on the comparison of element labels. Labels comparison may include syntactic and semantic metrics for determining the accuracy between labels. Moreover, techniques for computing the string edit distance, such as the Hamming distance [3], the Levenshtein distance [4, 5], or the Damerau-Levenshtein distance [6] can be used. Each of these metrics is defined as a minimal number of operations needed to transform one string into the other using deletion, insertion, substitution of a single character, or transposition of two adjacent characters.

Also, a business process model can be transformed to a graph or a net. Therefore, process models can be compared as graphs by applying the graph-edit distance metric [7] (structural matching).

The behavioral matching is an approach, based on comparing the behavioral components of models. An algorithm based on causal footprints was suggested in [8]. A causal footprint provides a definition of a set of conditions on the order of activities that hold for the model.

Our approach is based on the fact that process models, which need to be compared, should be represented in XML format. Although this approach is described and implemented for process models represented in BPMN XML 2.0, it can be extended to compare process models defined using other XML formats due to the hierarchical nature of XML.

Note that we didn't find any special tool for comparison of two XML files in accordance with their XML schema.

II. STRUCTURE OF XML SCHEMA

The structure of XML schema is a key factor for understanding the comparison algorithm proposed. In this section we will discuss the structure of XML schema by an example of the BPMN 2.0 XML schema format [9].

XML schema defines elements contained by an XML document and their types. Fig. 2 shows that BPMN 2.0 XML schema is represented by a list of elements descriptions and their complex (compound) and simple types.

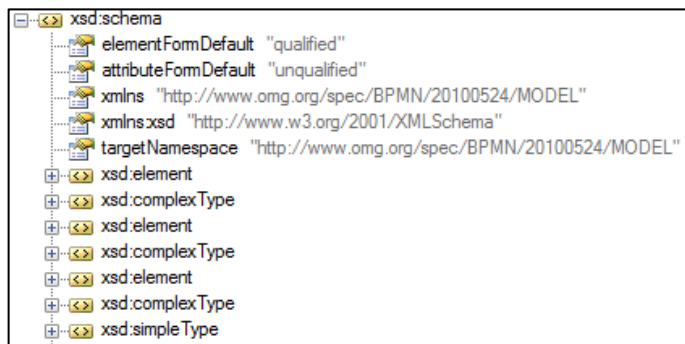


Figure 2. BPMN 2.0 XML schema

Let us consider a description of the element «subProcess» (Fig. 3).

```
<xsd:element name="subProcess"
            type="tSubProcess"
            substitutionGroup="flowElement"
/>
```

Figure 3. «subProcess» BPMN 2.0 XML element

Subprocesses in terms of BPMN represent multiple tasks that work together to achieve certain goals. The composite nature of subprocesses is reflected in a corresponding complex XML type (Fig. 4).

```
<xsd:complexType name="tSubProcess">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element
          ref="laneSet"
          minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element
          ref="flowElement"
          minOccurs="0"
          maxOccurs="unbounded"/>
        <xsd:element
          ref="artifact"
          minOccurs="0"
          maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute
        name="triggeredByEvent"
        type="xsd:boolean"
        default="false"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Figure4. «SubProcess» BPMN 2.0 XML type

The type «SubProcess» extends an abstract type «tActivity» with sets of lanes (containers used to logically organize activities within a subprocess), flow elements, which represent

all the elements contained, and artifacts, which stand for the comments to subprocess elements. Attributes «minOccurs» and «maxOccurs», indicating the minimum and maximum number of occurrences of an element, show that each inner element can be presented zero or more times within a subprocess. Thus, to compare subprocesses we need recursively compare all the contained elements.

The other element to be considered is a sequence flow (Fig. 5). Sequence flows are usually depicted as directed arcs and used to show the order, in which activities will be performed within a process. For each sequence flow identifiers of the source and the target nodes are specified using attributes of a special IDREF type. This should be taken into account during the comparison. Sequence flows and other connecting elements should be compared according to their source and target nodes, but not according to the identifiers of these nodes. In other words, two sequence flows coincide if their source and target nodes coincide, while nodes identifiers usually differ. This fact distinguishes our algorithm from other XML comparison algorithms, which don't consider element references.

```
<xsd:element name="sequenceFlow"
            type="tSequenceFlow"
            substitutionGroup="flowElement"/>
<xsd:complexType name="tSequenceFlow">
  <xsd:complexContent>
    <xsd:extension base="tFlowElement">
      <xsd:sequence>
        <xsd:element name="conditionExpression"
          type="tExpression"
          minOccurs="0"
          maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="sourceRef"
        type="xsd:IDREF" use="required"/>
      <xsd:attribute name="targetRef"
        type="xsd:IDREF" use="required"/>
      <xsd:attribute name="isImmediate"
        type="xsd:boolean" use="optional"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Figure 5. «sequenceFlow» element and «tSequenceFlow» type

Another important fact that should be taken into account is that XML schema contains abstract elements. Abstract elements are unavailable for end users, but used for inheritance. Their main purpose is to make language more extensible and allow adding new elements inheriting some parameters from their parents.

III. COMPARISON ALGORITHM

Now let us turn to the description of the comparison algorithm. First we have to define the notion of equivalent elements. Two XML elements are equivalent if and only if:

- they have the same names;
- for each attribute of the first XML element there exists one and only attribute of the second XML element, which has the same name and the same value and vice versa; Note that for IDREF attributes corresponding linked XML elements must be equivalent;

- for each nested element of the first XML element there exists one and only one equivalent nested element of the second XML element and vice versa.

First let us impose restrictions on the structure of XML documents. Assume that elements with IDREF attributes don't have nested elements; assume also that there are no IDREF links to these elements from other XML elements. Note that these restrictions are justified for XML documents, containing information on hierarchical process structure (e.g. subprocesses) and sequence flows connecting arbitrary process nodes. The algorithm consists of three steps:

A. The first step

The first step includes generation of a set of elements that are directly nested in the root element «definitions» for each model (Fig. 6).

```
<definitions xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="1"
targetNamespace="http://www.bizagi.com/definitions/1"
xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL">
```

Figure 6. XML element «definitions»

B. The second step

Now we have two sets of BPMN elements for two models at the first level. For each element from the first set we perform the following steps:

- select all elements with same name from the second set;
- if no elements were selected add an «error» message to the result of comparison;
- set the correspondence between the element from the first set and each selected element if:
 - they don't have nested elements and IDREF attributes, but they have the same sets of attributes with coinciding names and values;
 - there are correspondences between their nested elements and attributes, which can be obtained recursively using Step B.

If there are remaining elements from the second set with no corresponding elements add an «error» message to the result of comparison

C. The third step

Consider all the elements with IDREF attributes for both models.

- set the correspondence relation between them if and only if linked XML elements are in correspondence relations and not-IDREF attributes coincide as well;
- remove redundant correspondences, which are not supported by IDREF attributes.

This algorithm assists in determining equivalent elements, but generally speaking there is no guarantee that equivalence

relations will be constructed if multiple corresponding elements can be obtained for some element.

The algorithm was extended with an ability to specify relevant and non-relevant attributes.

The result of the comparison can consist of three types of messages, which describe main information about comparison:

- «error» - an error message;
- «warning» - an alert message;
- «info» - an information message.

A message takes an «error» status if the algorithm cannot find an equal element in another model. If for some reasons the algorithm cannot compare the non-relevant attributes of elements, a message should be added to a «warnings» list. A message should be added to an information list, if an element from the first model has more than one equal element from the other model.

IV. IMPLEMENTATION

After the structure of the XML schema is analyzed, the BPMN XML schema can be disassembled and transformed into an object-oriented model, which is implemented using some programming language.

We have developed our algorithms on the basis of ProM framework [10]. The ProM framework is a free open source product developed by the Eindhoven University of Technology. The algorithm for comparison two business process models in the BPMN 2.0 XML format was successfully implemented in ProM and can be used by business process analysts. Further, the main steps for applying a ProM plugin for comparing process models are shown.

Importing resources

First, the following resources should be imported to ProM:

- Model1.bpmn - the first business process Model
- Model2.bpmn - the second business process Model
- Schema.xsd – BPMN XML schema

After importing, these resources are displayed in the «Workspace» tab (Fig. 7).

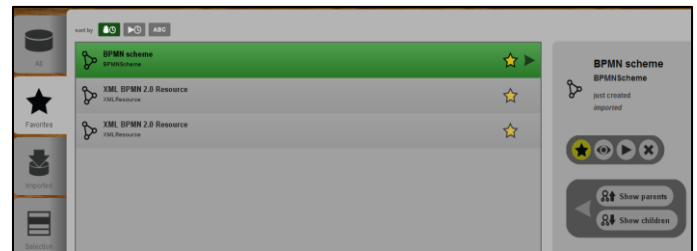


Figure 7. List of imported resources

Selecting and applying plugin

After importing resources the user selects a necessary plugin from the plugin list in the «Actions» tab. «XML BPMN 2.0 Comparator» plugin should be selected in our case (Fig. 8).



Figure 8. Selection of the «XML BPMN 2.0 Comparator» plugin

Analysis of the results

The results of the plugin's work are represented in an information window with the results which are divided into three groups: «error», «warning», «info» on the «Views» tab (Fig. 9).

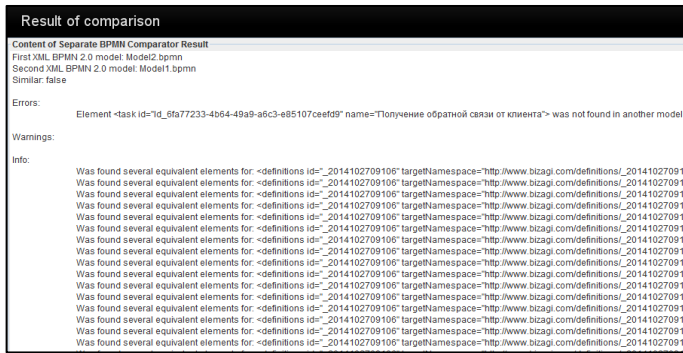


Figure 9. The result of the comparison of two models in the XML BPMN 2.0 format

The final report with results can be exported from the ProM in .txt and .html formats.

V. EXAMPLE

Suppose we have a shopping process model (Fig. 10). This model includes start, end events and the following tasks: checking order information, saving an order to database, receiving of payment, delivering the goods. The delivery service is responsible for delivering an order. Delivering an order is a subprocess, which includes the following steps: collect order, test order, pack order, and deliver order. After a model is discovered from an event log, there is a need to compare the real process model of e-shop (Fig. 10) with a reference process model (Fig. 11). These models should be imported to ProM framework and compared with «XML BPMN 2.0 Comparator» plugin.

As a result plugin reported that an element with type «Task» and name «Testing» in the subprocess «Delivery service» was not found in a reference model. Also, a complete list of attributes, which were not found the document starting from the root element, was produced. According to the comparison results, analysts can find errors, modify and improve process of organization.

VI. CONCLUSION

Nowadays, system and business analysts face a problem of process models comparison due to the changes in processes occurring under influence of various factors. Therefore, there is a real demand for tools capable to compare process models.

This paper introduces a novel approach for process models comparison, which uses their XML representations.

We have proposed an algorithm that can be used to compare process models in XML format. This algorithm was described by the example of BPMN 2.0 XML format. The BPMN format was chosen as the most popular format for modeling business processes.

The results of the research were successfully implemented in the ProM framework and can be further used by business process analysts.

ACKNOWLEDGMENT

This study was carried out within the National Research University Higher School of Economics' Academic Fund and is supported by Russian Fund for Basic Research (project 15-37-21103).

REFERENCES

- [1] Stephen A. White. Introduction to BPMN [Online]. Available: http://www.omg.org/bpmn/Documents/Introduction_to_BP_MN.pdf
- [2] W. M. P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, Springer-Verlag, Berlin, Germany, 2011.
- [3] D.Sanko and J. Kruskal, Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, Addison-Wesley, 1983.
- [4] V. Levenshtein, Binary codes capable of correcting spurious insertions and deletions of ones. Problems of Information Transmission, 1965, pp. 1-17.
- [5] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady, pp. 10-707, 1966. Original in Russian in Doklady Akademii Nauk SSSR, 1965, pp. 163-848.
- [6] F. Damerau. A technique for computer detection and correction of spelling errors. Comm. of the ACM, 1964, pp. 7-176.
- [7] Xinbo Gao, Bing Xiao, Dacheng Tao, Xuelong Li, "A survey of graph edit distance" in Pattern Analysis and Applications, vol. 13, 2010, pp. 113-129.
- [8] B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst, "Structural Patterns for Soundness of Business Process Models" in EDOC 2006 – International Enterprise Distributed Object Computing Conference, Hong Kong, 2006, pp. 116-128.
- [9] Object Management Group, "BPMN 2.0," [Online]. Available: <http://www.omg.org/spec/BPMN/2.0/>
- [10] Process Mining Group, Eindhoven Technical University, "ProM 6," [Online]. Available: <http://www.promtools.org/>

APPENDIX A. FIGURES

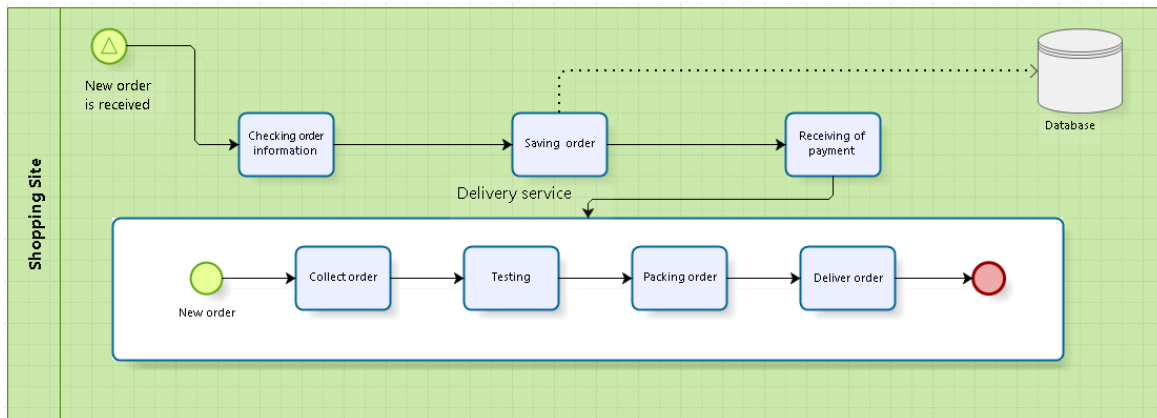


Figure 10. A real shopping process model

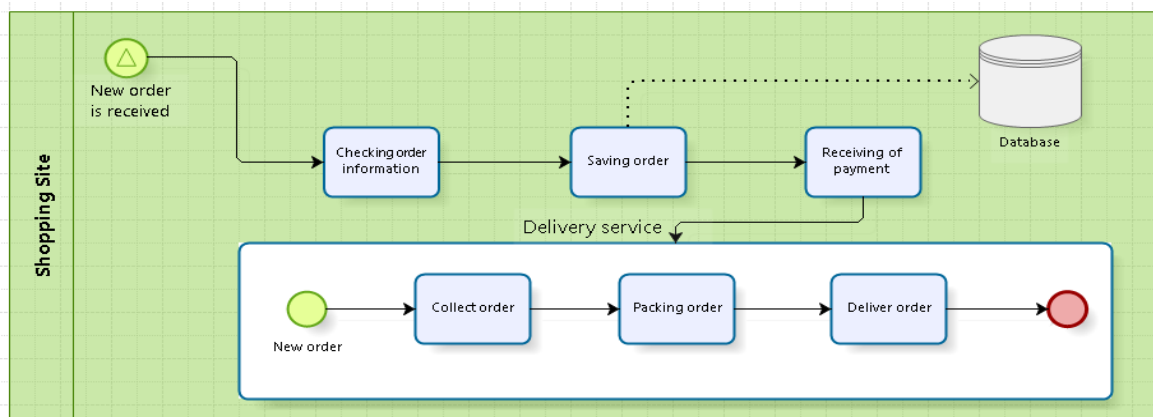


Figure 11. A reference shopping process model

Developing of a complex of software tools for organization and support of distance learning game system «3Ducation»

L.S. Zelenko

Software Systems Department
Samara State Aerospace University
Samara, Russia
Zelenko.larisa.s@gmail.com

A.E. Semenov

Software Systems Department
Samara State Aerospace University
Samara, Russia
alexandr.semenov.smr@gmail.com

D.A. Konopelkin

Software Systems Department
Samara State Aerospace University
Samara, Russia
dekanszn@gmail.com

M.A. Savachaev

Software Systems Department
Samara State Aerospace University
Samara, Russia
msavachaev@gmail.com

V.S. Ivanov

Software Systems Department
Samara State Aerospace University
Samara, Russia
arietis27@gmail.com

E.E. Poberezkin

Software Systems Department
Samara State Aerospace University
Samara, Russia
efim@poberezkin.ru

A.O. Grigoriev

Software Systems Department
Samara State Aerospace University
Samara, Russia
edspawn@gmail.com

The article describes the structure of distance learning system «3Ducation», the functions and capabilities of all its constituent software (components) as well as technologies and development tools of the system.

E-learning, gaming approach, technology of virtual reality, three-dimensional space, a web application, game engine Unity3D, database

I. INTRODUCTION

Currently distance education (e-learning) is becoming increasingly popular, almost all educational institutions present their courses electronically and provide access to them online. Virtual educational systems present a relatively new kind of learning systems, which combines the features of traditional systems of training, e-learning environments and achievements in information technology. The e-learning environment is generally understood as "system-organized set of means of communication, information resources, communication protocols, hardware and software and organizational methods, designed to meet the educational needs of users" [1]. Virtual learning environments provide comprehensive methodological and technological support for distance educational process, including training, management of the educational process and its quality.

Currently there are a lot of virtual and distance learning environments, but nevertheless there's a relevant task of creating virtual environments which use modern information technology, such as virtual reality technologies that make the educational space more interesting and learning process more fun. Social studies indicate that the boundary between the virtual and real worlds is being erased. The advantages of the three-dimensional virtual space are derived from human perception of information. Up to 80% of the information about the world a person receives through sight which works more effective when the world it sees is more imaginative. Teachers know that a simple and obvious example is often more effective than strict theoretical calculations. The most popular educational resources on the Internet (eg, Khan Academy [2]) increasingly rely on video instead of text.

Distance learning system «3Ducation» is built on two principles:

1) *game approach*, which aims to increase the interest of students by introducing interactive and continuous feedback, encouragement for achievements, teamwork capabilities and the presence of a competitive element to the system.

2) *virtual reality* involves the transfer of the learning process into three-dimensional environment that allows

you to remove the problems of the supply of educational material. This allows you to maintain and even increase the interest in self-learning, and thus enhances the effectiveness of training.

Combining the possibilities of advanced information technologies with teaching potential, it is possible to build an individual educational path for each student, taking into account his needs and features of information perception and processing.

II. SYSTEM ARCHITECTURE

Distance gaming learning system «3Ducation», developed at the Department of Software Systems of SSAU, is based on client-server technology and is built on the three tiered architecture (Fig. 1).

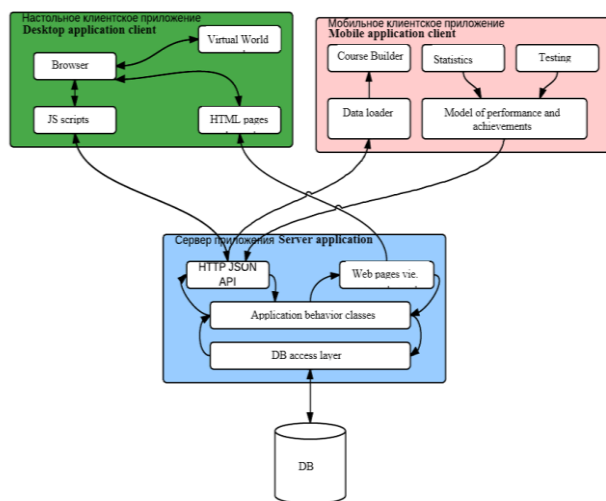


Fig. 1. System architecture

Server component of the system includes the server application and the database. The server application allows you to use the same logic in both desktop and mobile client.

The *client part* of the system is simply a web browser, which is used to view pages on the server (user only needs to install a small plugin Unity Web Player). 3D-scenes of the virtual world are integrated into the HTML-page, so the student can move through the virtual space as through the pages of the usual websites.

The *server part* of the system implements the MVC (Model-View-Controller) architecture, which defines three levels:

- 1) level of presentation of portal's web pages;
- 2) level of business logic and data access;
- 3) data level.

The *mobile client application* provides all the basic functions of the basic version of the system.

The network protocol TCP/IP is used as a network communication protocol. Controllers of behavior logic group serve the pages of the presentation group. The main component of the model (data level) is a database context; there is given a listing of all the essential classes included in the model, and all the controllers work with the database through it.

III. SOFTWARE FOR THE ORGANIZATION AND SUPPORT OF THE SYSTEM

System «3Ducation» has a complex structure (Fig. 2), it consists of a number of subsystems, each of which solves the problem of providing support for the system and its interaction with other systems. Let's take a closer look at the server side of the system.

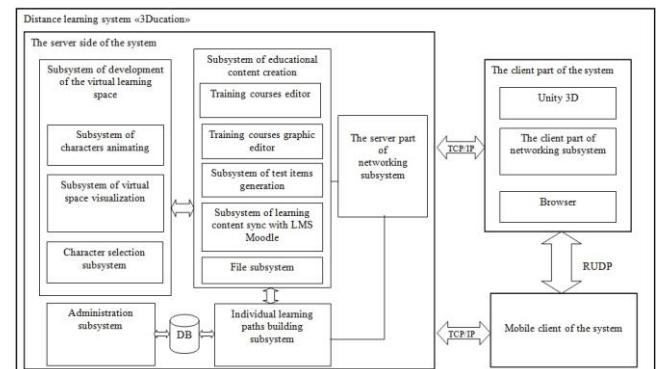


Fig. 2. Structure of system

A. Subsystem of development of virtual learning space

Virtual learning space consists of two parts: a permanent part and a dynamic one. Permanent part is presented in the form of the hall and includes a place of choice of the course from the list of available courses, as well as background information about the developers, the department and the university. The dynamic part is a set of connected rooms/corridors and is generated automatically based on the structure of the selected course and rooms templates, which are filled with specific content.

Subsystem of development of virtual learning space includes:

- 1) *subsystem of selection and configuration of characters*, which is designed to allow the user to select and customize the appearance of the avatar, which he will control in the virtual world;
- 2) *subsystem of animating characters*, which is designed to provide a realistic behavior of the selected characters;
- 3) *visualization subsystem*, which is responsible for the visualization of all the three-dimensional models of virtual learning space.

B. Subsystem of educational content creation

This subsystem includes:

- 1) *editor of the training courses*, which provides the teacher interface to edit, add or remove any element of learning content (lectures, test). Editor interface is shown in Fig. 3;
- 2) *graphical editor of educational courses* - software tool for building individual trajectory of a student. The editor allows to connect same level learning materials, such as topics or lectures and courses (Fig. 4);
- 3) *subsystem of test tasks generation*. Teacher, working with the module through the administrative part of the website, can add new test tasks or change the settings of the existing

templates of tests, which will then be available for students in the virtual three-dimensional space;

- 4) *subsystem of synchronization* of educational content with LMS Moodle, which is used to convert tests and lectures from the database of distance learning system LCMS Moodle to the database of learning system «3Ducation»
- 5) file subsystem.

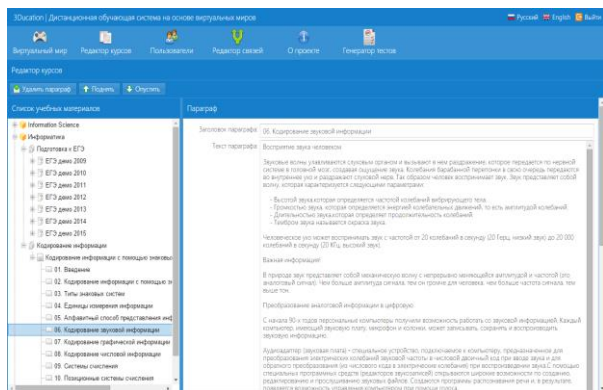


Fig. 3. Interface of educational content editor

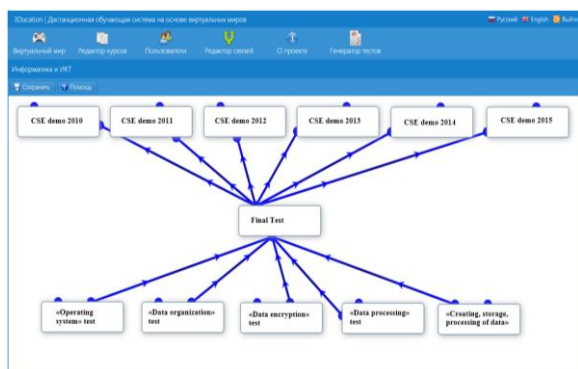


Fig. 4. Graphical editor of the course

C. Subsystem of administration

Distance gaming learning system «3Ducation» is a part of the information space of the school of computer science at SSAU. The other system is responsible for user registration and storage of their personal data. Therefore, the administration subsystem includes a subsystem that allows authentication using OpenID technology, which allows to use one account for multiple systems. In addition, the administration subsystem includes means to ensure the relevance of systems that allow the administrator to edit the information posted on the website of the system, using the user-friendly web interface instead of working with the source code of the system or its database. These software tools make it easier to maintain the system, improve the reliability of its work, ensure the confidentiality of stored data, and allow to maintain students' interest in learning.

D. Software providing network communication for mobile client on Android and Windows Phone

Currently the system «3Ducation» is implemented as a multi-user educational environment where students could work together to perform learning tasks, cooperating and communicating with each other, including using a mobile version of the system.

Development of multi-user mode required changing and/or adding the following operating modes of the system:

- support for joint passing of chosen course of study,
- joint passing of test tasks in cooperative, competitive and team modes,
- calculation of statistics of the learning process,
- possibility of communication between the participants.

During the development of the network part of the system the following main problems, inherent in mobile devices, have arisen and the following ways have been found to solve them:

1) device may have an unstable Internet connection: connection quality depends on many factors: signal strength, connection speed, the type of connection (Wi-Fi, 4G, 3G, Edge or GPRS). Solution: to use RUDP protocol for transmission of most data.

2) device can forcibly limit Internet connection: mobile devices are powered by batteries and have a small battery life. To increase this time, the OS developers and device manufacturers try to limit the consumption of one of the most "voracious" components – radio module. Solution: add mechanisms to suspend learning when connection is lost.

3) the device can easily change the IP-address: if device uses the Internet via a cellular network, IP-address of the device depends on the base station of operator, which leads to the fact that when the reception conditions are poor or when the user moves it changes very often. A similar situation occurs when connecting/disconnecting the Wi-Fi network. Solution: to not take into account the IP-address of the user, for identification only use cookies and xsrf-token.

IV. TECHNOLOGICAL SUPPORT OF EDUCATIONAL PROCESS

Distance learning system «3Ducation» extensively use capabilities of virtual reality technology (Virtual Reality) or virtual worlds. The criterion for selecting the underlying technology was the possibility to integrate virtual worlds into the browser that would ensure the integrity of the system. After careful analysis the free version of the game "engine" Unity3D was chosen. Its creators (the company Unity Technologies [4]) describe it as "the most powerful free game engine". Level of graphical effects of Unity3D is superior to both O3D and X3D graphics, but much more valuable fact is its simplicity, convenience and stability. Graphic editor allows to quickly model the geometry of the scene, without having to write code. To import any resource it is enough to just move the appropriate file in the project folder. The big advantage of Unity3D is an impressive collection of ready resources - household items and character models with a ready and highly customizable code responsible for controls and movement of the camera. By using Unity3D engine system can be developed quickly and in full, avoiding non-obvious problems that can slow down or stop the work.

E. Software development tools

Software selected to develop the system includes the following technologies [5]:

- development environment Microsoft Visual Studio 2010 and programming language C #;
- technology of web application development ASP.NET 4.0;
- framework ASP.NET MVC Frame-work 3.0;
- data access technology Entity Framework 4.0;
- database management system Microsoft SQL Server 2008;
- server software IIS 7.5;
- JavaScript-library ExtJS 4.0;
- development environment Unity Editor 3.4;
- three-dimensional graphics editor Blender 2.6.

F. Data storage and manipulation technologies

One of the main functions of the system is processing and storage of data, as well as correct display of it when generating the virtual world. For these purposes the data access technology Entity Framework is used. It allows to automatically generate a database and all tables on the basis of essential classes created by developer and populate them with the original data, if it was determined. This technology monitors all changes, made during the development of system, on the code level and, if necessary, modifies the structure of the database. The choice of Entity Framework determined selection of DBMS: Microsoft SQL Server 2008 is also a part of family of technologies from Microsoft and ensures the correct work of the above functions better than other options. The data necessary for the operation of the system «3Ducation» is stored in the database. In addition, part of the data is stored on the server in the form of files.

V. TEAM DEVELOPMENT OF THE SYSTEM USING GIT-REPOSITORY

The system «3Ducation» is being developed by a large team of developers, which obliges to use a version control system. After a comparative analysis of systems of this class version control system GIT has been selected, because it has the following advantages:

- decentralization (the presence of a local repository containing full information on all changes, allows to maintain full local version control and "fill" in the master repository only fully authenticated changes);
- good support of non-linear development;
- efficient operation of large projects;
- high performance and speed;
- reliable system of audit comparisons and data validation based on the hashing algorithm SHA1 (Secure Hash Algorithm 1);

- extensibility and configurability (there is a large number of graphical shells, which allow to quickly and accurately work with Git) [6, 7].

One of the extensions used in the repository is a simplified git-flow diagram (a general version of the diagram is shown in Fig. 5), which consists of master, develop and features branches. According to it the system «3Ducation» is being developed in several branches:

- branch, which always contains only release versions,
- branch, which stores the code between new releases,
- a set of branches, each of which is reserved for only one development feature.

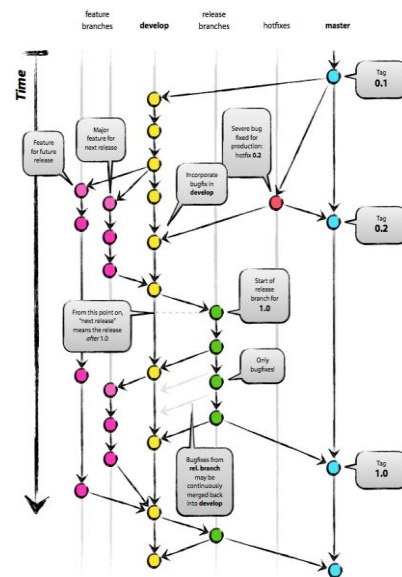


Fig. 5. General git-flow diagram

Thus, the use of the version control system Git allowed to clearly organize the work of the development team to synchronize the development process and increase the reliability of the system.

VI. CONCLUSION

Distance learning system «3Ducation» is designed for middle and high school students learning basic course "Computer Science". The system provides a unified interactive way to access information resources for both a teacher and a student, it can help to increase the effectiveness of the acquisition of knowledge and skills (both individual and social).

REFERENCES

- [1] The concept of creation and development of a unified system of distance education in Russia [Electronic resource] – URL: http://www.e-joe.ru/sod/97/2_97/st064.html.
- [2] The official website of Khan Academy [Electronic resource] – URL: <http://www.khanacademy.org>.
- [3] Zelenko, L.S. Virtual reality and game approach as a basis for constructing a three-dimensional learning space [Text] / L.S. Zelenko// Materials of the VIII International scientific and practical conference "Innovation in the development of informational communication

technologies (INFO-2012)"/ under. ed. of V.G. Domrachev, S.U. Uvaysov. - M.: MIEM, 2012. - P. 56-59.

[4] The official website of Unity3D [Electronic resource] –URL: <http://unity3d.com/company/>.

[5] Zelenko, L.S. Principles of the development of virtual learning system «3Ducation» [Text] / L.S. Zelenko, D.A. Zagumennov // Collection of selected works of the VII International scientific and practical conference "Modern information technologies and IT education". Under ed. of prof. V.A. Sukhomlina. - M.: INTUIT.RU, 2012. - P. 326-333.

[6] About - Git [Electronic resource]. - <http://git-scm.com/about>.

[7] Review of version control systems [Electronic resource]. - http://all-ht.ru/inf/prog/p_0_1.html.

Combined Classifier for Website Messages Filtration

Veniamin Tarasov
Povolzhskiy State University
of Telecommunications
and Informatics
Samara, Russia
Email: tarasov-vn@psuti.ru

Ekaterina Mezenceva
Povolzhskiy State University
of Telecommunications
and Informatics
Samara, Russia
Email: katya-mem@mail.ru

Danila Karbaev
Povolzhskiy State University
of Telecommunications
and Informatics
Samara, Russia
Email: danila@karbaev.com

Abstract—The paper describes a new approach to website messages filtration using combined classifier. Information security standards for the internet resources require user data protection however the increasing volume of spam messages in interactive sections of websites poses a special problem. Unlike many email filtering solutions the proposed approach is based on the effective combination of Bayes and Fisher methods, which allows us to build accurate and stable spam filter. In this paper we consider the organization of combined classifier according to determined optimization criteria based on statistical methods, probability calculations and decision rules.

Keywords—combined classifier; spam filter; optimization criterion

I. INTRODUCTION

The constantly growing volumes of data, number of uses as well as groups devoted to various subjects significantly decrease the effectiveness and the authenticity of communicated information. In this regard the task of increasing the efficiency of statistical data filtration and authentication algorithms becomes undoubtedly topical. The history of this subject in computer science accounts for more than 20-30 years and the trend is becoming more urgent. We can say that right now the antispam features of interactive sections of websites rest in the very initial stage of development.

The subject of message filtration in emails is widely developing, manual antispam methods are being used, and the issue of automated antispam protection of corporate websites becomes a priority on the agenda (including comments, forums and other interactive sections). In practice there are no universal software solutions to protect all types of interactive website sections from spam. There are only small number of specialized tools which prevent automatic messages posting. Some of them are designed for a particular content management system, such as WordPress in form of plugins: Akismet, Quiz, Spam Karma etc. These modules have some disadvantages: the distribution model “as is” do not include the statistical base, most of online services do not provide multilingual filtration and are limited only by the support of the English language. The other blog comment hosting services such as IntenseDebate, Disqus, Livefyre do not provide self-hosted option, except Discourse.

Thereby the spam filtering software solution should have the following properties: the use of multiple filtering methods, both formal and linguistic, united by a common intellectual decision making core; high speed and precision of the method; easy installation and use.

This work describes a new approach to spam filtration involving the combined use of Bayes and Fischer methods, allowing to significantly reduce the number of false triggering and increase spam detection.

II. CALCULATION OF COMBINED PROBABILITIES OF CONDITIONS

The main idea of message classification is based on selection of all conditions, calculation of probabilities of select conditions, and further combination of all calculated probabilities into one value for the studied message. Messages with a large number of spam attributes and little non-spam attributes will have a value close to 1, and the messages with a large number of non-spam attributes and little number of spam attributes will gain a value close to 0.

We will build a classifier of messages received by the website to grade the incoming messages into three categories (spam, non-spam, unidentified). In this respect, we need to identify all conditions (words and word combinations) in the message to be analyzed, calculate statistical probabilities for some select conditions and combine all probabilities into one value for the whole message. In most cases the probability of assigning a message to a certain category is a lot higher than to others, which results in further grading of such message.

Before calculating the combined probabilities of conditions, we need to calculate the probability of assigning a certain condition to a specific category. For this we can divide the identified number of messages with condition i in this category by the total number of messages in the same category, but we would rather use another method described below.

Let's assume:

F_{ai} is the number of messages with condition i in the spam group;

F_{bi} is the number of messages with condition i in non-spam group.

Then the statistical probability of appearance of i in a spam message can be calculated as follows:

$$p_{ai} = \frac{F_{ai}}{F_{ai} + F_{bi}}, \quad (1)$$

and the probability of appearance of i condition in a non-spam message, as follows:

$$p_{bi} = \frac{F_{bi}}{F_{ai} + F_{bi}}. \quad (2)$$

Thus, the number of messages with condition i in one category will be divided by the total number of messages featuring this condition i .

The use of (1) and (2) takes into account the fact that with time the number of messages in both categories may be equal, i.e. these formulas do not depend on the number of messages in a specific category.

Note that formulas above give accurate result only to those conditions, which filter is used in both categories. As the result the spam filter becomes too sensitive on early stages of learning applying to rare words. To solve this problem we need to calculate new probability with expected a priori probability (P_{ex}) and applied weight (w), then according to (1) and (2) add calculated probabilities.

If the probability $P_{ex} = 0.5$ and the weight of expected probability equals to one word ($w = 1$), we estimate weighted probabilities using (1) and (2):

$$\begin{aligned} \overline{p_{ai}} &= \frac{(w * P_{ex}) + p_{ai} * (F_{ai} + F_{bi})}{w + F_{ai} + F_{bi}}, \\ \overline{p_{bi}} &= \frac{(w * P_{ex}) + p_{bi} * (F_{ai} + F_{bi})}{w + F_{ai} + F_{bi}}. \end{aligned}$$

This approach allows to avoid division by zero in the following formulas and to take into account rare words.

To obtain combined probabilities of the whole document (message) we will use the dictionary, which is built on the step of filter learning. We introduce the following events: A – document is spam, B – document is non-spam. We assume that the probabilities are independent, thus the multiplication is allowed:

$$P(A) = \overline{p_{a1}} \times \overline{p_{a2}} \times \dots \times \overline{p_{aM}}, \quad (3)$$

- for the probability of words co-occurrence in spam;

$$P(B) = \overline{p_{b1}} \times \overline{p_{b2}} \times \dots \times \overline{p_{bM}} \quad (4)$$

- for the probability of words co-occurrence in non-spam [1].

III. DECISION RULES BASED ON BAYES THEOREM

To estimate the probability that word belongs to one of three categories (spam, non-spam, unidentified messages) we consider the two methods of classification. In this case we apply Bayes formulas using a priori knowledge [1].

We introduce two hypotheses for any given message:

H_A if the message is a spam,

H_B if the message is a non-spam.

Further, we introduce the following notation:

F_a is the total quantity of spam messages;

F_b is the total quantity of non-spam messages;

$p_a = \frac{F_a}{F_a + F_b}$ is a priori probability that a message is a spam;

$p_b = \frac{F_b}{F_a + F_b}$ is a priori probability that a message is not a spam;

$O_a = \frac{P_a}{1 - P_a}$ is a priori expectations that a message will be a spam;

$O_b = \frac{P_b}{1 - P_b}$ is a priori expectations that a message will be a non-spam.

Then basing on Bayes theorem using a priori knowledge we obtain:

$P(H_A) = \frac{P(A) \times O_a}{P(A) \times O_a + P(B) \times O_b}$ - a posteriori probability that a message is a spam;

$P(H_B) = \frac{P(B) \times O_b}{P(A) \times O_a + P(B) \times O_b}$ - a posteriori probability that a message is non-spam.

The probabilities $P(A)$ and $P(B)$ are estimated according to (3) and (4).

Given algorithm is implemented in spam detection and filtering system for websites. [2].

IV. DECISION RULES BASED ON FISHER'S METHOD

According to Fisher method all probabilities are multiplied together in a similar manner to Bayes method, then the natural logarithm is taken of the product and the result is multiplied by -2. To do this we introduce variable $hisqv$, which is estimated by the following expressions:

$$hisqv = -2 * \ln(P(A)) \text{ or } hisqv = -2 * \ln(P(B)),$$

where probabilities $P(A)$ and $P(B)$ are calculated according to (3) and (4).

Fisher proved that if the set of independent and random probabilities (3) and (4) is given, the value $-2 * \ln(P(A))$ follows the distribution of χ^2 with $2n$ degrees of freedom (n – the number of words in the document):

$$F(x) = \int_0^x \frac{t^{n-1} e^{-t/2}}{2^n \Gamma(n)} dt, \quad (5)$$

where $\Gamma(n)$ is the gamma function.

In view of foregoing using a representation of the gamma function of even argument (5) can be written as:

$$F(x) = \frac{1}{2^n (n-1)!} \int_0^x t^{n-1} e^{-t/2} dt \mid x = hisqv. \quad (6)$$

The calculation of the factorial and the integrand in (6) could cause the overflow error due to floating point numbers range in PHP programming language. Thus the recurrence formula is used in the calculation algorithm. Calculation the probability of (6) is implemented by Gaussian quadrature formula with 15 nodes:

$$\int_a^b f(t) dt \approx \frac{b-a}{2} \sum_{i=1}^n A_i f(t_i),$$

where $t_i = (b+a)/2 + (b-a)x_i/2$, and x_i are the nodes of Gaussian quadrature formula;

A_i are the Gaussian coefficients, ($i = 1, 2, \dots, 15$) [3]. In our case $a = 0$, $b = hisqv$.

The value returned by the function $F(hisqv)$ is low if a text contains many spam conditions. We need the opposite result to rate the message correctly. For this purpose we subtract the value from 1. The use of this subtraction for a large number of non-spam conditions allows us to get the probability that message is not spam.

However the Fisher method is not symmetrical. We need to combine the probabilities of spam and non-spam into a single value in the range between 0 and 1. For this we use the Fisher index:

$$I = \frac{1 + P(H'_A) - P(H'_B)}{2}, \text{ where:}$$

$P(H'_A) = 1 - F(-2 \ln(P(A)))$ is the probability that a document belongs to spam;

$P(H'_B) = 1 - F(-2 \ln(P(B)))$ is the probability that a document belongs to non-spam [4].

V. OPTIMIZATION CRITERIA FOR GRADING MESSAGES BASED ON STATISTICAL METHODS

Let's assume that all set of conditions is divided into classes A and B , where A – class of spam messages, and B – class of non-spam messages. The task of assigning a message to any of these classes is not directly connected to the statistical verification of the following hypotheses: simple hypothesis $H_A: X \in A$ against the alternative $H_B: X \in B$, where X is the message qualifying condition. As we know from the math statistics, if a message appertains to class A and it was qualified as class B , it will result in 1st type error with the conditional probability of α - level of importance. It will be an error of the alternative hypothesis selection H_B instead of the correct H_A . If H_B hypothesis is fair but, nevertheless, H_A was selected, the 2nd type error will occur with the conditional probability of β .

The 1st type error or false-negative error occurs if the spam filter erroneously leaks an undesired message through identifying it as non-spam (spam leakage or insufficient method completeness). Whilst the spam filter is capable of identifying a large share of undesired messages, the task of minimizing the number of faulty filtering of desired (non-spam) messages may become a higher priority, i.e. the task of 2nd type of error minimization.

The 2nd type error or false-negative error occurs if the spam filter erroneously classifies a legitimate message as spam (faulty triggering or method accuracy). The spam filter will be efficient with a lower number of such errors, i.e. with minimal 2nd type error level. However currently all antis spam systems demonstrate correlation between 1st and 2nd type errors.

The classifiers normally admit the compromise between the acceptable level of 1st and 2nd type errors, and use the threshold values for decision-making, which may vary. This results in the "strictness" or "softness" of the classifier. The level of significance set during the statistical hypothesis verification is taken as the threshold value. Whereas, the increase of the filter sensitivity leads to the increased occurrence of 1st type errors (spam leaks), and decrease of sensitivity – to increased occurrence of 2st type of error (false triggering).

VI. BAYES OPTIMIZATION CRITERION

We need to consider the losses related to 1st and 2nd type errors for evaluating the classification quality. For this we need to split the space of condition X into two semispaces X_A and X_B with point x_0 . Let's define c_1 as the conditional price of 1st type error and c_2 – conditional price of 2nd type error, $P(A)$ – a priori probability of A class, $P(B)$ – a priori probability of class B , $P(A) + P(B) = 1$. The values c_1 and c_2 depend on the price matrix coefficients $C_{2 \times 2} = \{c_{ij}\}$ and on the 1st and 2nd type errors:

$$c_1 = c_{12} \alpha + c_{11} (1 - \alpha), \quad (7)$$

$$c_2 = c_{21} \beta + c_{22} (1 - \beta). \quad (8)$$

These values are also called conditional risks with proven fairness of hypotheses H_A and H_B , respectively.

According to the decision making theory, we introduce the decision rule of classification, which minimizes the function of losses (risk) [3]:

$$R = c_1 P(A) + c_2 P(B). \quad (9)$$

where c_1 and c_2 are determined by (7) and (8).

Function (9) represents the average risk, which depends on the threshold value x_0 , because the values c_1 and c_2 depend on the x_0 value through type I and type II errors, therefore these errors are correlated.

Minimum value R_{min} of risk function (9) at the point x_0 is called Bayes risk.

$$\frac{f_1(X)}{f_2(X)} = \frac{c_{21} - c_{22}}{c_{12} - c_{11}} \cdot \frac{P(B)}{P(A)}, \quad (10)$$

where $f_1(X)$ and $f_2(X)$ are the probability density distributions of X condition on A and B classes respectively.

The right part in (10)

$$\frac{c_{21} - c_{22}}{c_{12} - c_{11}} \cdot \frac{P(B)}{P(A)}$$

is called likelihood ratio, which is

constant for the selection of c_{ij} . Thus, if the inequality $\frac{f_1(X)}{f_2(X)} > \frac{c_{21} - c_{22}}{c_{12} - c_{11}} \cdot \frac{P(B)}{P(A)}$ is true, the observable vector X

is related to A class; if the inequality

$$\frac{f_1(X)}{f_2(X)} < \frac{c_{21} - c_{22}}{c_{12} - c_{11}} \cdot \frac{P(B)}{P(A)}$$

is true, then observable

vector X is related to B class. If the equality

$$\frac{f_1(X)}{f_2(X)} = \frac{c_{21} - c_{22}}{c_{12} - c_{11}} \cdot \frac{P(B)}{P(A)}$$

is true, the observed vector X

is related to one of the classes A or B . The latter expression is the equation for the boundaries of A and B classes. This decision rule is related to Bayes rules [5].

The technique can be applied to many practical problems formulated in terms of statistical decision making theory with assumption that probability densities $f_1(X)$ and $f_2(X)$ are known. In most practical cases functions $f_1(X)$ and $f_2(X)$ are not known, and we need to determine estimations $\tilde{f}_1(X)$, $\tilde{f}_2(X)$ on training sets using approximation method [5], which can cause the classifier to slow down. Considering this fact we use the following approach: on the stage of filter learning the estimations $\tilde{f}_1(X)$, $\tilde{f}_2(X)$ are determined on small training sets of 100-200 elements, and the

optimality criterion to get such estimations can be excluded excluded from the program flow.

Results of numerous tests on training selections allowed identifying optimal threshold values for decision-making:

$x_H = 0,95$ for higher threshold and $x_L = 0,4$ for lower threshold.

Thereby we set strict limits for spam and regular for non-spam messages. Such threshold values provide minimum leakage of desired messaged into spam, i.e. minimum false triggering. However, it's notable that any system administrator will be able to easily set more convenient threshold values to suit his needs.

VII. COMBINED FILTER

In order to receive more valid results of spam detection we need to analyze multitudes of results of various filters and a subset of their overlaps.

We suggest exactly this kind of approach to classifier organization, which presumes the combined use of Bayes and Fischer methods for improved the filtration quality based on the analysis of subsets and set overlaps identified by both methods (spam, non-spam, false triggering and spam leaks).

Let's assume $S = \{s_i\}$ ($i=1 \div M$) – multitude of documents (messages), including both desired and spam messages; $S_B \subset S$ and $S_F \subset S$ – multitude of documents, identified by Bayes and Fischer classifiers, respectively. Then the subset resulting from the overlap $S_B \cap S_F$ against all indicated categories may be used for evaluating the quality of the combined filter operation (see Fig. 1).

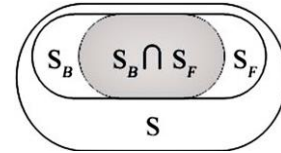


Fig. 1. Illustration of overlap degree of two subsets S_B and S_F .

The completeness of such overlap $S_B \cap S_F$ will also grade the subsets $S_B \setminus S_F$ and $S_F \setminus S_B$. As a measure of overlap degree of two sets S_B and S_F we suggest to use the absolute measure $N(S_B \cap S_F)$ – number of shared documents in these subsets. Thus, the maximum value of measure of l category (spam, non-spam, false triggering and spam leaks) will be used as the optimality criterion for spam filter self-teaching evaluation:

$$N_l(S_B^l \cap S_F^l) \rightarrow \max.$$

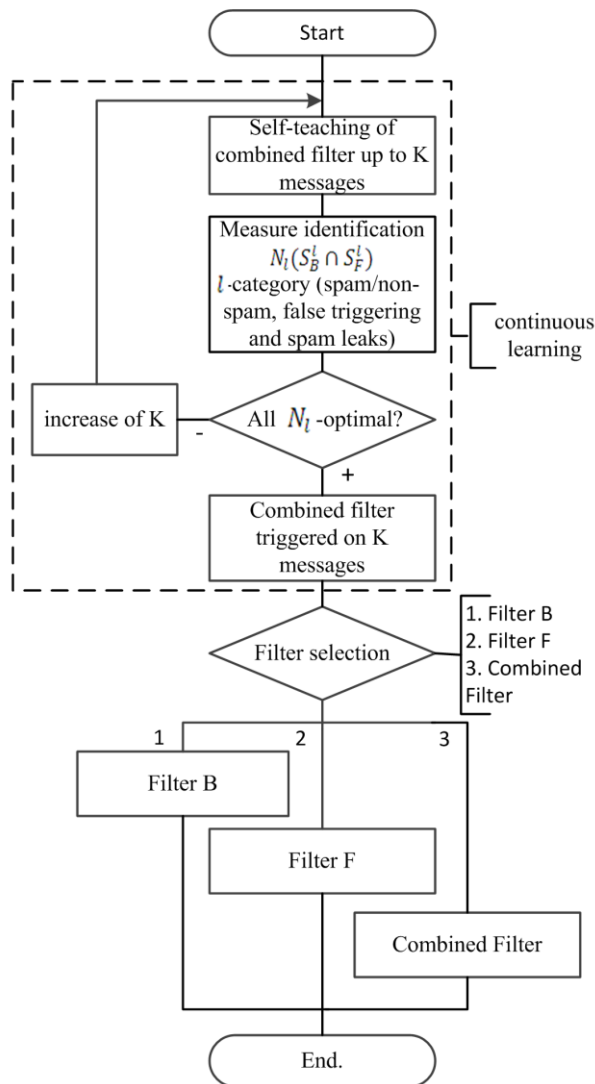
Once the best values of sets S_B and S_F overlap are reached across all categories, the administrator will be able to choose a filter for further application (see Fig. 2).

As a benefit of the combined filter implementation the evaluation of all components of the overall picture became possible:

- spam messages caught by both filters;
- spam filters caught only by Bayes or only Fischer filters;
- simultaneous false triggering of both filters;
- false triggering of each individual filter;

- simultaneous spam leaks by both filters;
- spam leaks of each individual filter.

we will be able to make a reasonable comparison of the combined filter self-teaching quality.



- [1] E. Mezenceva, V. Tarasov, "Securing computer networks. The method of multi-module spam filtering on websites," Information Technologies, vol. 6, pp. 18-22, 2012 (in Russian).
- [2] E. Mezenceva, "The software system of recognition and spam filtering on the sites," Certificate of state registration of the computer program №2011619160, [Registered in the Computer Program Registry, Moscow, on November 25th, 2011] (in Russian).
- [3] S. Nikolskiy, Quadrature Formulas. "Nauka", Moscow, 1974 (in Russian).
- [4] E. Mezenceva, V. Tarasov, "Computer networks security. Web programming of the multi-module spam filter," Software Engineering, vol. 4, pp. 27-32, 2012 (in Russian).
- [5] E. Mezenceva, V. Tarasov. "An optimal filter construction based on combining statistical classifiers," Information and communications technologies, book 1, vol. 4, pp. 53-57, 2013 (in Russian).

Fig. 2. The algorithm of combined filter accuracy evaluation

Before testing filter was trained on 1100 messages (400 spam and 500 non-spam). The tests were run on the flow of 1223 messages. The Bayes method showed 2.9 percent of the false triggering, 9.8 percent of spam omission. The Fisher method showed 1.5 and 4.5 percent accordingly. The combined filter showed the best result with 1.0 and 4.5 percent.

The experimental results confirmed the feasibility of using the selected filtering algorithms. Only having a whole picture,

Statistical data handling program of Wireshark analyzer and incoming traffic research

Veniamin Tarasov

Volga Region State University of
Telecommunications and Informatics
Moskovskoe sh. 77, Samara, Russia
Email: tarasov-vn@psuti.ru

Gleb Gorelov

Volga Region State University of
Telecommunications and Informatics
Moskovskoe sh. 77, Samara, Russia
Email: gleb_fox@bk.ru

Sergey Malakhov

Volga Region State University of
Telecommunications and Informatics
Moskovskoe sh. 77, Samara, Russia
Email: malakhov-sv@psuti.ru

Abstract— The paper presents a plugin to the Wireshark traffic analyzer to calculate the moments of the random variable – the interval between packets of incoming traffic. The article also presents the analytical solution for the average waiting time for a QS type $H_2/M/1$. Here H_2 is the 2nd order hyperexponential distribution law of the input flow time intervals. The final result is obtained as a solution of Lindley's integral equation using the method of spectral decomposition. It is shown that in this case the distribution laws of intervals between input flow requirements can be approximated at the level of their three first moments. The joint use of these results allows to fully analyze the incoming traffic by queuing methods.

Keywords— traffic analyzer, wireshark program, numerical characteristics of random variables, Lindleys equation, method of spectral decomposition

I. INTRODUCTION

The identification of the distribution laws of intervals is particularly sophisticated problem, at the same time the traffic as a random process tends to be constantly changing. It is known, the queuing theory is based on the laws of distribution of intervals between income and service requirements. Therefore it is important to know the numerical characteristics of these intervals or their moments. In this paper we propose to use the Wireshark analyzer to determine such characteristics [1].

II. DESCRIPTION OF THE PROGRAM WIRESHARK

Wireshark (previously, Ethereal) is a traffic analyzer for Ethernet computer networking technology and some others. In June 2006 the project was renamed Wireshark due to trademark issues [1].

The functionality provided by Wireshark is very similar to the capabilities of the tcpdump program, but Wireshark has a graphical user interface and additional features for sorting and filtering information. The program allows the user to view all the traffic through the network in real time, shifting the network card to promiscuous mode. (Eng. Promiscuous mode) (Fig. 1).

Wireshark is an application that can display the structure of a wide variety of network protocols, and therefore allows parsing network packets, showing the value of each field protocol at any level. The use of Pcap packet capture library allows capturing data only from those networks that are supported by this library. However, Wireshark can work with

multiple formats of input data an open data files captured by other programs that enhances the capture.

The features include:

- deep analysis of hundreds of protocols, with the regular addition of new ones;
 - capturing network traffic in real time, followed by analysis at any time;
 - standard three-pane packet browser (standard package has three regions);
 - cross-platform: there are versions for most types of UNIX, including Linux, Solaris, FreeBSD, NetBSD, OpenBSD, Mac OS X, as well as for Windows;
 - The captured from network information can be viewed by using the graphical user interface or by using the TTY-mode utility TShark;
 - the most powerful sorting and filtering in the industry;
 - a great opportunity to VoIP analysis;
 - read / Write a large number of file formats capture: tcpdump (libpcap), Pcap NG, Catapult DCT2000, Cisco Secure IDS iplog, Microsoft Network Monitor, Network General Sniffer® (compressed and uncompressed), Sniffer® Pro, and NetXray®, Network Instruments Observer, NetScreen snoop, Novell LANalyzer, RAD-COM WAN / LAN Analyzer, Shomiti / Finisar Surveyor, Tektronix K12xx, Visual Net-works Visual UpTime, WildPackets EtherPeek / TokenPeek / AiroPeek, and many other;
 - capture files that compressed with gzip can be unpacked immediately;
 - capturing real-time data can be effected via Ethernet, IEEE 802.11, PPP / HDLC, ATM, Bluetooth, USB, Token Ring, Frame Relay, FDDI, and the other (depending on the platform);
 - decoding support for many protocols, including IPsec, ISAKMP, Kerberos, SNMPv3, SSL / TLS, WEP, and WPA / WPA2;
- Highlighting rules can be applied to the package list for quick, in-intuitively analysis;
- output data can be exported to XML, PostScript®, CSV, or plain text.

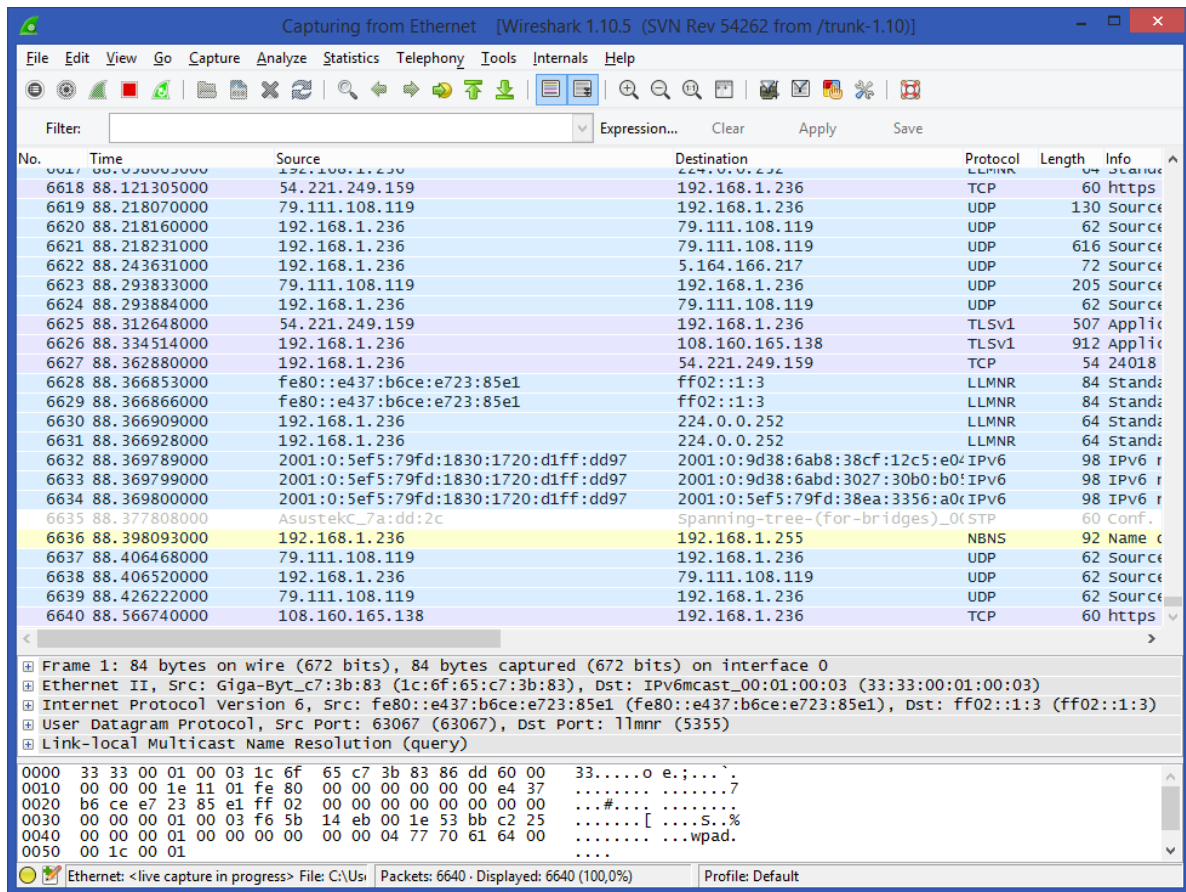


Fig. 1. The example of a network traffic capture by Wireshark

CSV is one of the formats of data export, convenient for viewing (Fig. 2). This file can be opened in any text editor or spreadsheet editor for analysis and calculation of performance.

However, it is difficult to process the data in case of intense traffic even in the spreadsheet editor. Furthermore the traffic data can be stored in more than one file. This article describes a software solution for the calculation of the numerical characteristics of packet arrival intervals. The main advantage of this analyzer is his work on a small scale of time (microseconds), in contrast to the same program NetFlow Analyzer, which captures packets-per-minute rate.

III. DETERMINATION OF THE MOMENTS OF THE INTERARRIVAL TIME OF INCOMING TRAFFIC

The program developed by the authors of the present paper allows, in addition to the analyzer, to retrieve the packet arrival times, isolated the incoming traffic from the entire data set received by Wireshark. Next, using the well-known formulas of mathematical statistics, it can be defined the moment characteristics of the timing. We use the statistics to the third order statistical properties, which provides representations of the distribution of the intervals.

For example, the coefficient of variation shows the difference from a Poisson traffic flow and with asymmetry

gives an indication of the degree of weight in the distribution tails.

The average value of the interval between adjacent packets

$$\bar{t} = \frac{1}{N} \sum_{k=0}^N (t_{k+1} - t_k),$$

where t_k – packet arrival times, N – the number of intervals analyzed.

$$\text{Custom dispersion } D = \overline{t^2} - \bar{t}^2,$$

where $\overline{t^2} = \frac{1}{N} \sum_{k=0}^N (t_{k+1} - t_k)^2$ – the second initial moment.

The coefficient of variation $c = \sigma / \bar{t}$, where $\sigma = \sqrt{D}$.

$$\text{Asymmetry } A_s = (\overline{t^3} - 3 \cdot \bar{t}^2 \cdot \bar{t} + 2 \bar{t}^3) / \sigma^3,$$

$$\text{where } \overline{t^3} = \frac{1}{N} \sum_{k=0}^N (t_{k+1} - t_k)^3.$$

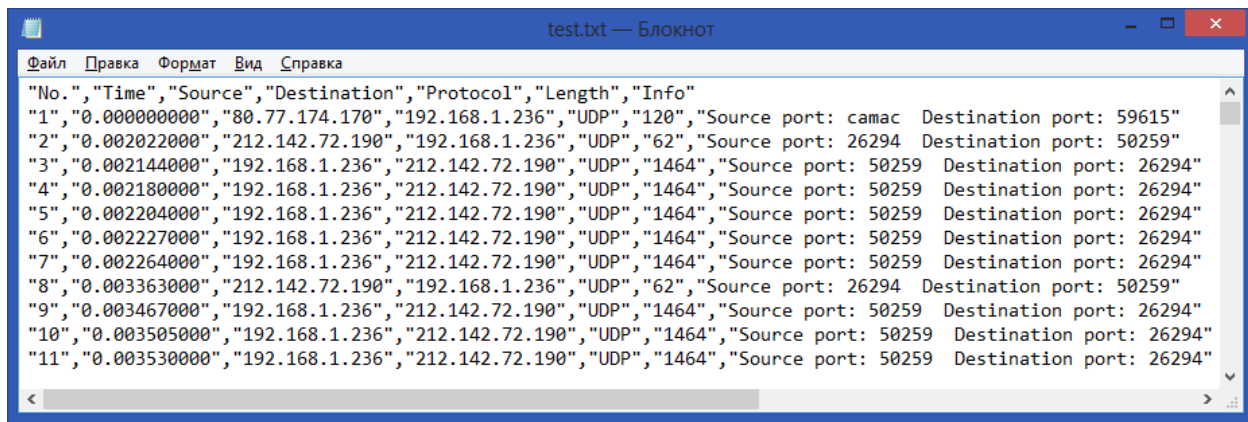


Fig. 2. The example of the data exported to the CSV format

If a large amount of data is divided into several blocks, then these formulas are determined by the average group, and then their mean values.

IV. TIME DATA ANALYSIS SOFTWARE AND RESULTS

To calculate the moments of the interval between adjacent packets, we developed a program, which selects only the data related to the inbound packet from the input file, containing the capture of a network traffic data, and calculates intervals and moments.

The features include:

- sample timing of the data packets arrived at said host;
- calculation of the time intervals between the incoming packets;
- calculation of the torque characteristics for intervals of received packets;
- saving time of the data packets arrived in binary and text format;
- saving data packet arrival intervals in binary and text formats;
- output and saving torque characteristics in a text format;

The program handles text files containing the data as shown in Fig. 2 or similar.

For the program the two classes (in terms of object-oriented programming) are developed:

- *TrafficLogParams* – stores the packet arrival time, their intervals and calculates the torque characteristics. Also provides the methods to store and download the data from files;
- *LogParser* – static class that produces an analysis of the input file and adds data to the *TrafficLogParams* class.

The input of *LogParser* main method is the file name and IP-address of the host. Each line of the source file is processed

and from the selected data on the time and two IP-address - the address of the sender and the recipient's address. If the recipient field matches the host IP-address, then the packet arrival time is added to the array such times in *TrafficLogParams* class.

```
public static TrafficLogParams
TextFileParser(string fileName, string ip, bool
isIncoming)
{
    TrafficLogParams log = new TrafficLogParams();
    StreamReader file = new StreamReader (fileName);
    string[] currentLine;
    int lineNumber = 0;
    int ipIndex;
    if (isIncoming)
        ipIndex = 2;
    else
        ipIndex = 1;
    while (!file.EndOfStream)
    {
        currentLine = GetDataArray
        (file.ReadLine().Trim());
        lineNumber++;
        try
        {
            if (MinimizeIp (currentLine[ipIndex]) ==
            MinimizeIp (ip))
            {
                log.AddTime(ParseDouble(currentLine
                [0]));
            }
        }
        catch (FormatException ex)
        {
            MessageBox.Show(string.Format("{0}\nСрок
            a = {1}", ex.Message, lineNumber));
        }
    }
    file.Close();
    return log;
}
```

The second most important method of LogParser splits the input string into elements, checking every element belonging to the format of time or IP-address, and returns them as an array.

```
private static string[] GetDataArray(string
input)
{
    string[] data = new string[3];
    string currentValue = "";
    int symbolIndex = 0;
    int valueIndex = 0;
    while (symbolIndex < input.Length && valueIndex
< 3)
    {
        while (symbolIndex < input.Length &&
(char.IsDigit(input[symbolIndex]) ||
IsSeparator(input[symbolIndex])))
        {
            currentValue += input[symbolIndex];
            symbolIndex++;
        }
        if (currentValue != "")
        {
            if ((IsDouble(currentValue) ||
IsIp(currentValue)))
            {
                data[valueIndex] = currentValue;
                valueIndex++;
            }
            currentValue = "";
            if (valueIndex >= 3)
            {
                symbolIndex = input.Length;
            }
        }
        while (symbolIndex < input.Length &&
!char.IsDigit(input[symbolIndex]) &&
!IsSeparator(input[symbolIndex]))
        {
            symbolIndex++;
        }
    }
    return data;
}
```

The method checks if the input symbol is a separator "." or ",",. Such testing is important only for the time data, as in some countries, the fractional part is separated by a comma (for example, in Russia), rather than a point. It is for the reason, when a string representation of a number is converted to its equivalent real number denoting the time, the standard method is not used programming language, and its modification depends on the regional settings.

```
private static double ParseDouble(string value)
{
    if (CultureInfo.CurrentCulture
.NumberFormat.NumberDecimalSeparator == ",")
    {
```

```
        value = value.Replace(',', '.');
    }
    else
    {
        value = value.Replace('.', ',');
    }
    return double.Parse(value);
}
```

When comparing the IP-address of the host with the IP-address on the current line of the log file to minimize the usual pro-IP-address to the general form. In other words, IP-address will be equal 010,014,000,011 10.14.0.11.

The program was used to analyze the data file of the traffic coming to the proxy server of the university with almost an hour-long data set. The input file contains more than 2150000 rows, which could not be processed manually. Were obtained the following results (Fig. 3):

File	Help
Initial moment of the 1st order:	5,097781e-003
Initial moment of the 2nd order:	3,325837e-004
Initial moment of the 3rd order:	5,505049e-005
Dispersion:	3,065963e-004
Variation coefficient:	3,434807e+000
Asymmetry:	1,025441e+001
Packets count:	628183
Ready!	

Fig. 3. The result of the analysis program log files

V. RESEARCH OF QUEUING SYSTEM $H_2/M/1$

The data indicate that the analyzed traffic differs from a Poisson (coefficient of variation $c = 3,43$ instead of 1), the asymmetry value $A_s = 10,25$ indicates that the distribution of intervals between the packets of traffic relates to a heavy-tailed distributions. For example, for Poisson flow of $A_s = 2$. The calculation of the characteristics of such traffic requires appropriate mathematical apparatus. For the analysis of such traffic the authors of [2] proposed the new results for the system $H_2/M/1$. We will describe the basic results from the article.

It is known, as example from [3], to study queuing systems (QS) $G/G/1$ the integral equation of Lindley is used:

$$W(y) = \begin{cases} \int_{-\infty}^y W(y-u) dC(u), & y \geq 0, \\ 0, & y < 0 \end{cases}, \quad (1)$$

where $W(y)$ is the probability distribution function (PDF), the waiting time in line requirements $C(u)$ is the PDF limiting random variable, $U = \lim_{n \rightarrow \infty} U_n = x_n - t_{n+1}$, and x_n is the time of the n -th service requirement C_n , and is the time interval between the t_{n+1} arrival of the requirements C_n and C_{n+1} .

To solve (1), a spectral method is used that reduces to using the expression $A^*(-s) \cdot B^*(s) - 1$ and finding a representation as a product of two factors, which would give a rational function of s [3]. Thus, to find the latency distribution, the following spectral decomposition is used:

$$A^*(-s) \cdot B^*(s) - 1 = \frac{\psi_+(s)}{\psi_-(s)} \quad (2)$$

where $\psi_+(s)$ and $\psi_-(s)$ are rational functions of s , which can be factored. The functions $\psi_+(s)$ and $\psi_-(s)$ must satisfy certain conditions [3]:

1. For $\text{Re}(s) > 0$, the function $\psi_+(s)$ is analytic without zeros in the half-plane;
 2. For $\text{Re}(s) < D$, the function $\psi_-(s)$ is analytic without zeros in the half-plane, (3)
- where D is a positive constant determined from the following condition:

$$\lim_{t \rightarrow \infty} \frac{a(t)}{e^{-Dt}} < \infty.$$

Moreover, the functions $\psi_+(s)$ and $\psi_-(s)$ must have the following properties:

$$\begin{aligned} \text{for } \text{Re}(s) > 0 \quad \lim_{|s| \rightarrow \infty} \frac{\psi_+(s)}{s} &= 1; \\ \text{for } \text{Re}(s) < D \quad \lim_{|s| \rightarrow \infty} \frac{\psi_-(s)}{s} &= -1. \end{aligned} \quad (4)$$

We know that all the main characteristics of Qs are derived from the average waiting time, and therefore all subsequent calculations will be performed with respect to the average waiting time in the queue requirements.

Consider QS $H_2/M/1$, where H_2 designates the hyperexponential distribution 2nd order arrival time requirements in a density function

$$a(t) = p\lambda_1 e^{-\lambda_1 t} + (1-p)\lambda_2 e^{-\lambda_2 t}, \quad (5)$$

and M – notation exponential law services with a density function

$$b(t) = \mu e^{-\mu t}. \quad (6)$$

The Laplace transform of (5) has the form

$$A^*(s) = p \frac{\lambda_1}{s + \lambda_1} + (1-p) \frac{\lambda_2}{s + \lambda_2}, \quad (7)$$

and function (6):

$$B^*(s) = \frac{\mu}{s + \mu}. \quad (8)$$

Now we define (2) for the distributions (5) and (6) from (7) and (8):

$$\begin{aligned} \frac{\psi_+(s)}{\psi_-(s)} &= \left[p \frac{\lambda_1}{\lambda_1 - s} + (1-p) \frac{\lambda_2}{\lambda_2 - s} \right] \frac{\mu}{\mu + s} - 1 = \\ &= \frac{[p\lambda_1(\lambda_2 - s) + (1-p)\lambda_2(\lambda_1 - s)] \cdot \mu - (\lambda_1 - s)(\lambda_2 - s)(\mu + s)}{(\lambda_1 - s)(\lambda_2 - s)(\mu + s)} = \quad (9) \\ &= \frac{\mu(a_0 - a_1 s) - (\lambda_1 - s)(\lambda_2 - s)(\mu + s)}{(\lambda_1 - s)(\lambda_2 - s)(\mu + s)}, \end{aligned}$$

where the coefficients $a_0 = \lambda_1 \lambda_2$, $a_1 = p\lambda_1 + (1-p)\lambda_2$.

The numerator of the right side of (9) is a third degree polynomial $s(s^2 - c_2 s - c_1)$, and it remains to determine the coefficients for the decomposition of the factors. The coefficients of the polynomial are:

$c_1 = \mu[\lambda_1(1-p) + \lambda_2 p] - \lambda_1 \lambda_2$, $c_2 = \lambda_1 + \lambda_2 - \mu$. Then the expression (9) can be factored:

$$\frac{\psi_+(s)}{\psi_-(s)} = \frac{s(s^2 - c_2 s - c_1)}{(s - \lambda_1)(\lambda_2 - s)(\mu + s)} = \frac{s(s + s_1)(s - s_2)}{(s - \lambda_1)(\lambda_2 - s)(\mu + s)},$$

where $-s_1 = -(\sqrt{c_2^2/4 + c_1} - c_2/2)$ is the negative root of the quadratic equation in the numerator, and is the $s_2 = \sqrt{c_2^2/4 + c_1} + c_2/2$ positive root.

Further, omitting some calculations, we obtain the Laplace transform of the density function of the waiting time:

$$W^*(s) = \frac{s_1(s + \mu)}{\mu(s + s_1)}. \text{ Hence } \frac{dW^*(s)}{ds} = \frac{s_1\mu(s_1 + s) - s_1(s + \mu)\mu}{\mu^2(s + s_1)^2}.$$

Using the properties of the Laplace transform, we find that the average waiting time is

$$\bar{W} = - \left. \frac{dW^*(s)}{ds} \right|_{s=0} = \frac{-s_1^2\mu + \mu^2 s_1}{\mu^2 s_1^2} = \frac{1}{s_1} - \frac{1}{\mu}. \text{ Finally, the}$$

average waiting time is

$$\bar{W} = \frac{1}{s_1} - \frac{1}{\mu}, \quad (10)$$

where $s_1 = \sqrt{c_2^2/4 + c_1} - c_2/2$, $c_1 = \mu[\lambda_1(1-p) + \lambda_2 p] - \lambda_1 \lambda_2$, $c_2 = \lambda_1 + \lambda_2 - \mu$.

VI. PRACTICAL USE OF THE RESULTS.

Consider the result (10) for example, the input distribution, with a heavy tail (fig. 3). Using the Laplace transform (7) we can determine the initial moments of the distribution (5):

$$\begin{cases} \bar{\tau}_\lambda = \frac{p}{\lambda_1} + \frac{(1-p)}{\lambda_2} \\ \bar{\tau}_\lambda^2 = \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} \\ \bar{\tau}_\lambda^3 = \frac{6p}{\lambda_1^3} + \frac{6(1-p)}{\lambda_2^3} \end{cases}$$

Next, substituting the results obtained in step 1 from the initial moments of the distribution of intervals between bursts to determine the unknown parameters of the input distribution (5): λ_1 , λ_2 and p , we obtain the following system of equations:

$$\begin{cases} \frac{p}{\lambda_1} + \frac{(1-p)}{\lambda_2} = 5.0978e - 003 \\ \frac{2p}{\lambda_1^2} + \frac{2(1-p)}{\lambda_2^2} = 3.3258e - 004 \\ \frac{6p}{\lambda_1^3} + \frac{6(1-p)}{\lambda_2^3} = 5.5050e - 005 \end{cases} \quad (11)$$

The solution of (11) in the package Mathcad yields the following results: $p \approx 0.950$, $\lambda_1 \approx 417.985$, $\lambda_2 \approx 17.556$.

In case of load of the channel equals to 0.4, intermediate parameters: $c_1 \approx 10999.4$; $c_2 \approx -54.655$, $s_1 \approx 135.707$ and the average waiting time $\bar{W} \approx 5.329 \cdot 10^{-3}$ s.

For comparison, let us look to the average waiting time for an M/M/1 system. In this case, the intensity of service equals to $\mu \approx 490.196$, and the channel loading $\rho = 0.4$.

Then the average waiting time of packets

$$\overline{W} = \frac{\rho/\mu}{1-\rho} = \frac{0.4/490.196}{1-0.4} = 1.36 \cdot 10^{-3} \text{ s.}$$

Thus the queuing model taking into account the distribution and its weight in the tail of the input, gives a delay about four times larger than the classical model.

CONCLUSION

This paper has presented how optimistic are the results given by classical M/M/1 system in comparison to the system in the case of high $H_2/M/1$ weightiness tail of the distribution of the input stream. Therefore, the approach can be successfully applied in the modern teletraffic theory where packet delays in the incoming traffic are significant.

Note that the distribution, which contains three unknown parameters λ_1 , λ_2 and p , allows to use the moment equations to approximate the unknown input distribution in the first three moments.

REFERENCES

- [1] Wireshark official web-site URL: <http://www.wireshark.org/> [02.02.2014]
- [2] Tarasov V.N., Bakhareva N.F., Gorelov G.A. Matematizheskaya model trafica s tyazhelohvostnym raspredeleniem na osnove sistemy massovogo obsluzhivaniya $H_2/M/1$. [Mathematical model of traffic from heavy-tailed distributions with based queuing system $H_2/M/1$]. Infocommunicationye tehnologii, 2014, no. 3, pp.36-41.
- [3] Kleinrock L. Queueing Theory. Tran. from English. edited by V.I. Neumann. M. Mechanical Engineer-ing, 1979.

Automatic Virtual Link Configuration for Simple AFDX Networks

Arthur Yalaletdinov, Alexey Khoroshilov
Institute for System Programming
Moscow, Alexander Solzhenitsyn st.,25.
Email: {yalaletdinov, khoroshilov}@ispras.ru

Abstract—The paper discusses a problem of configuring AFDX networks, which requires to find answers on several questions: how to distribute communication data streams by virtual links, how to route these virtual links via network switches and how to choose transmission parameters of virtual links, AFDX switches and end-systems. A simple solution is proposed that allows to automatically solve the problem for small-to-medium sized networks.

I. INTRODUCTION

Until recently, avionic systems have been built according to a federated architecture. Federated architectures have hundreds of wires and device-specific interfaces. It means that with the ever increasing number of embedded avionic functions, maintenance and analysis becomes critical. The Integrated Modular Avionics (IMA) is modern concept which replaces federated architecture of airborne platforms [1]. In a federated architecture, each system has private avionic resources, whereas in an IMA architecture avionic resources can be shared by several systems. Communication level of an IMA system based on Avionics Full Duplex Ethernet (AFDX) architecture. Processing units (called Core Processing Modules or CPM) of IMA system communicate using an AFDX network. AFDX is developed based on IEEE 802.3 standard (Ethernet) and it aims at providing reliable data communication for avionics applications [2].

The core idea of AFDX is a Virtual Link (VL). If make an analogy with federated architecture it replaces point-to-point wired data link but virtual link is point-to-multipoint unidirectional multiplexed connection. Only one End System within the Avionics network should be the source of any one VL. Each VL characterised by two parameters:

- Bandwidth Allocation Gap (BAG) is minimum time frame between two consecutive frames. The allowed BAG values calculate by formula $BAG = 2^k, k = 0, \dots, 7$.
- Maximum Frame Size (MFS) defines the maximum size of the transmitted AFDX frames

BAG and MFS give a regulation-mechanism for network bandwidth used by the given VL.

So, in section II we formulate problem statement. Then, section III presents an overview of related works in the problem sphere. In section IV we talk about simple VL routing method based on Breadth-first search algorithm. And finally, we make some conclusions about methods and approaches.

II. PROBLEM STATEMENT

So, we have a physically preconfigured AFDX network. In other words we have a set of CPMs $CPMS = \{cpm_1, \dots, cpm_n\}$, a set of switches $SW = \{sw_1, \dots, sw_l\}$ and a set of wired connections between CPMs and switches ports W . Moreover CPM and switch can be connected through only one link (one CPM port to one switch port), but for connecting two switches it is possible to use several ports of each of them.

Each CPM cpm_i defined by three parameters:

- *tick* - tick time (ms)
- *opTime* - atomic operation time (ticks)
- *maxUtil* maximum processor utilization

There is a set of partitions $PRTS = \{p_1, \dots, p_m\}$. For each one of them we know

- *period* - period of partition (ms)
- *duration* - duration of partition (ms)
- *mms* - maximum message size transmitted to destination partition without fragmentation, $17 \leq mms \leq 1471$ (bytes).
- $destds(p_j) = \{p_j^1, \dots, p_j^l\}$ - set of destination partitions

Further more it is important to keep in mind network latency requirements. So, delay between the transmission message by the source partition p_i and the reception by the destination p_j is upper bounded with $d(p_i, p_j)$. Let $D = \{d(p_i, p_j)\}$ the set of these delays.

Aggregation and segregation play role of additional constraints. In case of aggregation it means that two or more partitions must be placed on one CPM. Segregation is opposite. Let's denote $AC \subseteq P \times P$ - set of aggregation constraints. If $(p_i, p_j) \in AC$ - partitions p_i and p_j must be placed on some CPM, else - partitions p_i and p_j don't have that constraint. And similarly $SC \subseteq P \times P$ - set of segregation constraints. If $(p_i, p_j) \in SC$ - partitions p_i and p_j must be placed on different CPMs, else - partitions p_i and p_j don't have that constraint.

The problem is to distribute partitions on CPMs, generate set of virtual links VLS with their transmission parameters and route them satisfying all constraints such as aggregation, segregation, network latency requirements.

III. RELATED WORKS

The work [3] dedicated VLs defining and routing. Authors first show how to set the BAG and MFS parameters of a VL so as to minimize the reserved bandwidth while transmitting the data within their maximum delivery time. Next they consider the case where a source partition has to send multiple messages to the same set of receivers. And finally, proposed an exact mixed-integer-linear programming (MILP) formulation of the routing problem as to maximize the minimal residual capacity of the links.

Authors of work [5] claim that the weakness of previous approach is that the optimized parameters found in a single virtual link cannot be feasible when they are used in finding feasible configurations of multiple virtual links in an AFDX network switch and focus on finding feasible BAG and MTU parameters of VLs in an AFDX switch for a given VL of messages. In that paper proposed solution of this problem in a one AFDX switch.

Next, some authors in work [4] propose two heuristic algorithms to configure the number of VL in ADFX networks. By the proposed algorithms, two or more flows are grouped in one VL only if the required bandwidth for VL being merged flows is less than sum of bandwidth of merging flows.

In all these papers assumed that AFDX network physically preconfigured and partition distribution is defined.

The most of works are devoted to AFDX network analysis. Different methods and approaches has been proposed, such as Trajectory approach, Network Calculus, Response Time Analysis (RTA) [6], [7]. These methods allow to estimate transmission delay at the level of ES.

The paper [8] presents approach to optimizing setting of priorities of the AFDX traffic flows, with the objective to obtain tighter latency and queue-size deterministic bounds. Those bounds are calculated by Network Calculus method. Optimization method based on genetic algorithm.

Often for solving optimization problems are used genetic algorithms (GA) . We have an experience with applying this approach to AFDX network configuration problem. For an GA it is necessary to set a metric function called “Selection operator” that can estimate “quality” of generated population. So Network Calculus and Trajectory methods are a good choose to create that function. In our case the role of population was played by configuration of AFDX network (defined physical connections and VLs routes) [9].

So, really there are three approaches here:

- (mixed) integer linear programming
- greedy algorithm approaches
- heuristic approaches (such as genetic algorithms)

The first way’s success depends on quality and fullness of formulation the problem as a MILP problem. Greedy algorithm usually easy to design but needs a proof of correctness. Heuristic approaches such as genetic algorithms are hard to implement and require serious analysis and selection of options of implemented algorithm.

In all considered papers solution constructed by splitting the problem on the two independent parts. The first is defining partition distribution and the second is configuring and routing VLs. Partition distribution algorithms don’t take into consideration how partitions communicate. But it is necessary to use information about transmitted packet sizes and their periods simultaneously in both problems.

IV. ROUTING ALGORITHMS

Proposed algorithm shown in pseudo code 1.

First of all, we create partition distribution and try to build schedule for all CPMS. For this purpose we use the corresponding functionality of our MASIW tool set [10]. In the code below it is denoted as functions *DistributePartitions()* and *BuildSchedules()* (see details in[11]).

On the next step we distribute communication data streams by virtual links and defines its transmission parameters in *ConfigureVLs()* function. It is implemented following the ideas described in [3].

Algorithm 1 Configure and routing

```

counter = 0, VLS = {}
while counter < N do
    PD = null
    repeat
        PD = DistributePartitions(CPMS, PRTS, AC, SC, D)
        if BuildSchedules(PD) = true then
            break
        end if
    until PD ≠ {}
    if PD = {} then
        break
    end if
    VLS = ConfigureVLs(PD, D)
    //(here VLs transmission parameters are defined, but
    //VLs are not routed)
    for VL in VLS do
        RouteAndChoose(VL)
        //(here VL is updated with routing information)
    end for
    if NetworkAnalyse(CPMS, SW, W, VLS) = true then
        break
    end if
    counter = counter + 1
end while
return VLS

```

Then for each virtual link *RouteAndChoose()* function builds a route via AFDX network switches. It considers the network as a graph where vertexes are switch ports and edges are any connections between ports (even inside switch). The routing starts breadth-first search of shortest paths in the graph. It chooses a random shortest path for the first time, but if there is another virtual link that starts and ends in the same AFDX switch, it chooses another shortest path by Round-Robin. The final step is to check correctness of the network against network latency requirements, queue size limits, etc. That is implemented using existing functionality of static network analyzers of MASIW tool set.

V. CONCLUSION

The paper presents a work in progress that is aimed to automatically configure small-to-medium sized AFDX networks. We proposed a simple approach that was implemented within MASIW tool set, but its systematic evaluation is still an open problem.

Another direction for future research is to investigate more sophisticated routing algorithms that take into account load level of particular AFDX switches or ports. It should allow to avoid building invalid configurations, where some of AFDX switches are overloaded.

REFERENCES

- [1] Richard L. Alena, John P. Ossenfort IV, Kenneth I. Laws, Andre Goforth, Fernando Figueroa. Communications for Integrated Modular Avionics, Aerospace Conference, 2007 IEEE, ISSN :1095-323X
- [2] AIRCRAFT DATA NETWORK PART 7. AVIONICS FULL DUPLEX SWITCHED ETHERNET (AFDX) NETWORK, June 27, 2005
- [3] Ahmad Al Sheikh, Olivier Brun, Maxime Cheramy, Pierre-Emmanuel Hladik. Optimal Design of Virtual Links in AFDX Networks. 2012
- [4] YoungJun Cha and Ki-Il Kim. Heuristic Algorithm for Virtual Link Configuration in AFDX Networks. Department of Informatics, Engineering Research Institute, Gyeongsang National University, Jinju, 660-701, Korea
- [5] Dongha An, Hyun Wook Jeon, Kyong Hoon Kim, and Ki-Il Kim. A Feasible Configuration of AFDX Networks for Real-Time Flows in Avionics Systems. Department of Informatics Gyeongsang National University, Jinju, 660-701, South Korea
- [6] Tawk, M., Liu, X., Jian, L., Zhu, G. et al., Optimal Scheduling and Delay Analysis for AFDX End-Systems. SAE Technical Paper 2011-01-2751, 2011, doi:10.4271/2011-01-2751.
- [7] Henri Bauer, Jean-Luc Scharbarg, Christian Fraboul. Applying Trajectory approach to AFDX avionics network. Universite de Toulouse IRIT/ENSEEIH/INPT - 2, rue Camichel 31000 Toulouse, France
- [8] F. Frances, C. Fraboul, J. Grieru. Using Network Calculus to optimize the AFDX network
- [9] S.Shubin. Synthesis of architectural models of real-time control systems based on genetic algorithms, Technical report ISP RAS, 2013
- [10] Buzdalov D.V., Zelenov S.V., Kornychin E.V., Petrenko A.K., Strakh A.V., Ugnenko A.A., Khoroshilov A.V.. Tools for System Design of Integrated Modular Avionics, Proceedings of the Institute for System Programming of RAS, volume 26, 2014. Issue 1. pp. 201-230. DOI:10.15514/ISPRAS-2014-26(1)-6
- [11] Gruzdev A., Zelenov S. Sufficient condition for the impossibility of building schedule for real-time systems of strict periodicity, Technical report ISP RAS, 2015

Effective Use of Cloud Computing Resources in the Distributed Information Systems for Providing Quality Multimedia Services

Irina Bolodurina
Department of Applied Mathematics
Orenburg State University
Orenburg, Russia
prmat@mail.osu.ru

Denis Parfenov
Faculty of Distance Learning Technologies
Orenburg State University
Orenburg, Russia
fdot_it@mail.osu.ru

Abstract—Existing approaches to the use of cloud computing resources is not efficient. Modern multimedia services require significant computing power, which are not always available. In this paper, we introduce an approach that allows more efficient use of limited resources by dynamically scheduling the distribution of data flows at several levels: between the physical computing nodes, virtual machines, and multimedia applications.

Keywords—cloud computing, cloud system, computing node, computing resource, highload information systems, load balancing, quality of multimedia services, virtual machine, virtual resource component;

I. INTRODUCTION

The information flows between computing nodes in local and global networks has been steadily increasing each year. It is true not only for large data processing centers, but also for locally datacenters (DC) specializing in industry, economy, health and so on. An important area to use local DCs is education. Universities are increasingly using their own DCs to support integrated automated information systems (IAIS), providing end users with network multimedia services.

The need for more resources is one of the problems of high-loaded IAIS. The consumption of resources unlike the available volumes grows exponentially. [5]. The analysis of request flows to IAIS services shows their structure heterogeneity [1]. Modern IAIS services are based on the concept of cloud computing. However, the problem of limited resources used for cloud systems remains relevant [4].

The use of virtualization and cloud computing allows to consolidate several online services located on virtual machines (VM). It reduces the number of physical servers. But to effectively deploy applications on VM it is necessary to solve the problem of resource planning based on variable loads and service level agreement (SLA) [3]. The most flexible architecture of cloud computing is the infrastructure as a service (IaaS). This architecture allows the user to control a

pool of computing resources. This approach can imply the start of operating systems and applications, and the creation of virtual machines and networks. Thus, cloud computing leads to significant cost savings due to the increased load density [2].

However, the above is not enough to consolidate computing power, to reduce the infrastructure overheads and to reach optimal performance of cloud systems. To use the cloud infrastructure effectively new methods and algorithms should be developed to control components of cloud systems. It demands determining the formal structure of a cloud system [6].

II. MODEL OF RESOURCE VIRTUALIZATION OF CLOUD SYSTEMS

In our research, we have developed a model of computing resources of cloud systems. The conception of virtualization of computing resources is based on abstractions representing the tuples of relations between the interconnected elements of subsets.

The cloud system can be represented as a set of interconnected objects. They are computing nodes (*Snode*), system storages (*Sstg*), network attached storages (*Snas*) and scheduling servers (*Srasp*). The number of objects and the content of each set may vary depending on the cloud's size and its use.

Each compute node can run multiple instances of virtual machines represented as a set:

$$Snode_i = \{VM_{i,1}, VM_{i,2}, \dots, VM_{i,k}\}, \quad (1)$$

where k is the number of virtual machines on a compute node i , $i = 1 \dots l$ (l – number of nodes).

Each virtual machine belonging to the set (1) can support several applications and services represented as a set:

$$VM_j = \{App_{j,1}, App_{j,2}, \dots, App_{j,n}\}, \quad (2)$$

where n is the total number of applications and services, $j=1 \dots m$ (m - number of VMs).

The network attached storage includes a set of predefined VM images.

$$Snas_y = \{VMimg_{y,1}, VMimg_{y,2}, \dots, VMimg_{y,p}\}, \quad (2)$$

where $y = 1 \dots z$ (z - number of network attached storages).

Each VM image contains an operating system with preinstalled software and predetermined hardware parameters.

$$VMimg_{y,z} = \{OS_1, OS_2, \dots, OS_r\}, \quad (4)$$

The work of entire cloud system is performed using the planning system for certain operations defined by the scheduling servers.

$$Srasp = \{Rtask_1, Rtask_2, \dots, Rtask_f\}, \quad (5)$$

The distributed storage system usually consists of failover RAID arrays $Sstg_i = \{RDisik_1, RDisik_2, \dots, RDisik_d\}$ containing the information for multimedia services

$$RDisik_d = \{Data_1, Data_2, \dots, Data_s\}, \quad (6)$$

In addition, the cloud system also contains virtual and physical switches for interconnection between all the components in a network.

Each component of a cloud system $Shcn = \{Snode, Snas, Srasp, Sstg, VM \dots\}$ has the following characteristics:

$$Shcn = (State, Mem, Disk, Diskn, Core, Lan), \quad (7)$$

where $State \in \{\text{"on"}, \text{"off"}\}$ is the state of the component;

$Mem \in N$ is the size of RAM;

$Disk \in N$ is the disk capacity for storage;

$Diskn \in N$ is the number of storage devices;

$Core \in N$ is the number of processor cores;

$Lan \in N$ is maximum bandwidth of the network adapter;

The set of virtual machines can be divided into subsets $VMnode = \{Snode, Snas, Sstg, \dots\}$ to isolate computing resources for different services from each other.

The cloud system is a dynamic object changing at time t . Its state can be formalized in an oriented graph form:

$$Shcn(t) = (Node(t), Connect(t), App(t)), \quad (8)$$

where $Node(t) = \{Node_1, Node_2, \dots, Node_\lambda\}$ are active elements included in one of the sets $Snode_i, Sstg_j, Snas_k, Srasp_m$;

$Connect(t) = \{Connect_1, Connect_2, \dots, Connect_v\}$ are active connections by users to the virtualized applications;

$App(t) = \{App_1, App_2, \dots, App_n\}$ are active instances of applications running on virtual resources.

So we determine the structure of a cloud system and mechanisms of its component interaction. In such a system simultaneous servicing heterogeneous user requests is not trivial task.

To optimize the mechanism of access to information system resources it is necessary to analyze the main data flows transferred within the cloud system.

III. MODEL OF DATA FLOWS IN HIGHLOAD INFORMATION SYSTEMS BASED ON CLOUD COMPUTING

For flows analysis in our study, we used information systems of educational institutions. For analysis the most popular multimedia services have been determined. The research considered distance education systems (DES) consisting of different interactive applications.

In our research has built a level classification of applications:

- Level 1: The subsystem for monitoring the students' knowledge in real time;
- Level 2: The subsystem of the electronic library;
- Level 3: The subsystem of webcasts and webinars.

In our study, we have determined the general features of the use of the local DC's equipment.

- the load on the key resources is periodic and irregular;
- requests to multiple types of resources come at the same time;
- load distribution is not optimal, which results in loss of service at peak loads;
- up to 90% of the load is predetermined, as pre-registration is used for access to resources;
- up to 70% of the load arises due to multimedia educational resources.

Information flows at each level have their own characteristics. The intensity of servicing requested flows in the information system depends on the target application level. In a study we use the statistical analysis of the load on the most popular applications used in information systems of the university. Evaluation time for requests to various applications allow to forecast flows and ensure efficient allocation of resources. We using the goodness of fit chi-square Pearson to obtain data to test the hypothesis of distribution laws requests for incoming flow. In general, the intensity of incoming and service of a request flow for each class of applications is determined by the distribution function, which is described by the following distribution laws:

- for level 1 - Chi-squared distribution;
- for level 2 - Weibull distribution;
- for level 3 - Pareto distribution.

Flows of data transmitted in the IAIS are usually processed in several phases. At the same time in each phase several similar elements can be used providing balancing and load sharing between the components of the information system. The number of components in each phase depends on the functionality of the information system and the number of applications included in its composition. Suppose an information system has the form:

$$IS = \{S_1, \dots, S_r\} \quad (9)$$

where S_i - a component that performs data processing on the basis of the incoming flow of user requests, $i = 1..r$ (r - the total number of components of the information system). The

number of phases f in the flow path of user requests in an information system depends on its architecture.

The purpose of each phase according to its location in the processing sequence is:

- The first phase is the distribution of data flows between the LAIS resources in the cloud;
- The second phase is the dynamic scaling of the computing resources in the cloud;
- The third phase is data processing by user applications using storage systems and databases.

The components of the third phase include nodes of storage systems and database management systems for providing access to multimedia services in the cloud.

In detail the set of components of an information system is represented in form:

$$IS = \{S^1_l, \dots, S^1_n, S^2_l, \dots, S^2_m, S^3_l, \dots, S^3_k\}, \quad (10)$$

where S^j_i is the i component of the j phase;

$m \in N, n \in N, k \in N$ are the numbers of components included in the system for the respective phases f .

We also introduce the input components S^0_i which transmit data flows into an information system, and output components S^4_i receiving data flows from the cloud infrastructure. Consequently, the set describing the information system is transformed to:

$$IS = \{S^0_l, \dots, S^0_b, S^1_l, \dots, S^1_n, S^2_l, \dots, S^2_m, S^3_l, \dots, S^3_k, S^4_l, \dots, S^4_p\}, \quad (11)$$

where $p \in N, l \in N$ are the numbers of components in the input and output of cloud information system.

Each component S^j_i of the information system at any time can service multiple requests from different users. In the process of the user request data flows are generated upstream and downstream of the component. Their individual characteristics vary in time.

We designate all the incoming flows of component S^j_i as X^j_i , and the outgoing as Y^j_i , where i is the number of the components at the j service phase. Each request flow can be described as a set of characteristics. Suppose, there are l^j_i incoming flows and p^j_i outgoing flows for a component S^j_i .

Then for the incoming flow $v=1..l^j_i$, we introduce a set of characteristics:

$$X^{(j,v)}_i(t) = (x^{(j,v)}_{1,i}(t), \dots, x^{(j,v)}_{k,i}(t))^T \quad (12)$$

where

$x^{(j,v)}_{1,i}$ is the intensity of receiving requests in each incoming flow v of the component S^j_i ;

$x^{(j,v)}_{2,i}$ is the service time of the request flow v of the component S^j_i ;

$x^{(j,v)}_{3,i}$ is the intensity of servicing requests of the request flow v of the component S^j_i ;

$x^{(j,v)}_{4,i}$ is the service discipline of the flow v of S^j_i , which determines the order of service in accordance with the prioritization algorithm in the information system;

$x^{(j,v)}_{5,i}$ is the service class of the flow v of S^j_i ;

$x^{(j,v)}_{6,i}$ is the number of requests received from the flow v of S^j_i .

For outgoing flow $\mu=1..p^j_i$ of the component S^j_i the feature set includes:

$$Y^{(j,\mu)}_i(t) = (y^{(j,\mu)}_{1,i}(t), \dots, y^{(j,\mu)}_{k,i}(t))^T \quad (13)$$

The service path for each flow can be dynamically changed. The number of unique flows depends on the number of components in each phase.

A set of incoming flows at each phase j can be represented as:

$$X^j = \bigcup_{i=0}^{n_j} X^j_i \quad (14)$$

where j is the number of the service phases, n_j is the number of flows at phase j .

Consequently, all the incoming flows of the information system can be represented as:

$$X = \bigcup_{j=0}^f X^j \quad (15)$$

where f is the number of service phases.

For output flows the similar conditions are used :

$$Y^j = \bigcup_{i=0}^{n_j} Y^j_i \Rightarrow Y = \bigcup_{j=0}^f Y^j \quad (16)$$

To effectively serve user requests forming data flows in the information system, there must be an single-valued mapping of the form $R: X \rightarrow Y$.

In addition, for service of any request at each moment of time the matrix H of transitions between the phases of service is constructed depending on the class of the request and the current load of the system.

The graph of transitions between phases can be built using the function:

$$Y^{j-1}_e = R(X^{j,v}_i), \quad Y^{j-1}_e \in Y \quad (17)$$

where e is the component of phase $j-1$ directing data flow v to component S_i^j of phase j , $v=1..l_i^j$.

Then for any component S_i^j the set of all the input flows received from component S_i^{j-1} located in the previous phase is represented in the form:

$$X_i^{j,j-1} = R_j^{-1} [Y_i^{j-1} \cap R(X_i^j)] \quad (18)$$

where j is the phases of service.

Then effluents element S_i^j directed to the element S_i^{j+1} represented in the form:

$$Y_i^{j,j+1} = Y_i^j \cap R(X_i^{j+1}) \quad (19)$$

So $X^{j*} = \bigcup_{i=0}^n X_i^j$ and $Y^{j*} = \bigcup_{i=0}^m Y_i^j$ can describe the incoming and outgoing flows of phase j respectively.

In real systems, outgoing flows can overlap and get serviced on the same computing node that results in the formation of internal queues at each service phase.

To describe this process it is necessary to determine the connections between output flows of component S_i^j at phase j and all the components at phase $j+1$. Considering the above the set Y^{j*} becomes:

$$Y^{j*} = \bigcup_{S_i^j} \left[Y_i^{j,0} \bigcup \left(\bigcup_{S_i^{j+1}} Y_i^{j,j+1} \right) \right] \quad (20)$$

For a description of intersecting incoming flows within one phase two functions are introduced:

$$X^{j,j+1} = Q_x^j(Y^{j*}) \quad (21)$$

$$Y^{j,j+1} = Q_y^j(Y^{j*}) \quad (22)$$

where $Q_x^j(Y^{j*})$ characterizes input intersecting flows and $Q_y^j(Y^{j*})$ characterizes output intersecting flows for phase $j+1$.

Similarly, a set of input flows entering the phase of service can be defined. The flows of user requests can also intersect.

Consequently, an input data flow arriving on the component S_i^j at phase j from all the components at phase $j-1$ can be represented as:

$$X^{j*} = \bigcup_{S_i^j} \left[X_i^{j,0} \bigcup \left(\bigcup_{S_i^{j-1}} X_i^{j,j-1} \right) \right] \quad (23)$$

To describe the intersecting flows from the phase we introduce two functions:

$$X^{j,j-1} = P_x^j(X^{j*}) \quad (24)$$

$$Y^{j,j-1} = P_y^j(X^{j*}) \quad (25)$$

where $P_x^j(X^{j*})$ characterizes intersecting input flows, and $P_y^j(X^{j*})$ characterizes intersecting output flows from phase $j-1$.

Thus, the functions (21) and (25) describe the data flows between phases of service in an information system within a cloud.

To describe the whole multiphase information system we formalize the description of flows in each phase in the form $R^j : X^j \rightarrow Y^j$.

Thus data flows in an information system within a cloud can be represented as:

$$Y_i^j = R^j(X_i^j) = \begin{cases} R(X_i^j), & X_i^j \in X^j \\ P_y^j(X^{j*}), & X^{j*} \in \bigcup_{S_i^j} \left[X_i^{j,0} \bigcup \left(\bigcup_{S_i^{j-1}} X_i^{j,j-1} \right) \right] \\ Q_x^j(Y^{j*}), & Y^{j*} \in \bigcup_{S_i^j} \left[Y_i^{j,0} \bigcup \left(\bigcup_{S_i^{j+1}} Y_i^{j,j+1} \right) \right] \end{cases} \quad (26)$$

Data flows and their characteristics may change over time and our representation thereof should also include time t .

The description of an information system should include both internal and external factors so the parameter of external influence F should be introduced.

Then data flows in a cloud system can be described in the form:

$$Y_i^j = R^j(X_i^j, t, F) \quad (27)$$

IV. CLOUD SYSTEM VIRTUAL RESOURCES CONTROL ALGORITHM

The above models allow to determine the most appropriate computing nodes of the information system and the virtual machines that contain the required instances of multimedia applications. The control system should provide uninterrupted user service and effective virtual resource control in case of limited physical resources.

The main task of the control system is scheduling of computing resources at each moment of time. For highload information systems effective scheduling is important because the load on the services may vary greatly within short time intervals. In a cloud system there is a need to plan resource consumption optimally to prevent resource exhaustion for the application already running.

As distinct from other information systems the flow of user requests in the educational environment is predictable due to the subscriptions for multimedia services. The control algorithm for user access to virtual information resources consists of two interconnected processes.

One of these processes is scheduling. The scheduling algorithm collects data on the incoming requests and classifies them by the levels determined with the priorities of applications for business processes. The input data for the

algorithm are the applications described according to the template that includes a virtual machine image with the given configuration of hardware and software and user session characteristics.

Based on this template and data analysis of connections the algorithm calculates the configuration to deploy the required service. In the case of identical sets of VM software the already stored images are used. To optimize the use of computing resources the algorithm generates three variants of virtual machine configurations.

The first variant provides reserve performance in the case of unexpected increase in the number of users. The scaling factor in this case is calculated dynamically.

The second variant provides a predetermined low performance of virtual machines for the given number of users. This approach is most effective for small special purpose user groups. It allows to reduce the overhead in case few working users, the number of subscribers being large.

The third variant uses user-predetermined characteristics, including a fixed number of running instances of virtual machines regardless of the number of users. In this case the algorithm is only used to limit the computing resources. It calculates the maximum number of virtual machines that are available in the configuration selected by the user.

The second process within the algorithm is direct service of user requests and resource scaling during the work of applications. The algorithm considers the total number of requests from each source which allows to predict the load on the running applications within the cloud. Then the algorithm migrates virtual machines between computing nodes based on the collected data in accordance with a predetermined plan, thereby scaling the work of applications.

For efficient use of resources within the above processes, additional instances of virtual machines are created in the online storage of images for support the applications providing an access for the minimum amount of users.

In the case of predicted load increase on a certain service, the algorithm deploys a full image of the media resource and analyzes the incoming user requests. If the load does not exceed the number of queries in an ordinary flow, the algorithm switches the load to the appropriate image and turns off the virtual machine.

The scheme of an integrated approach to optimization using cloud computing, is presented in figure 2.

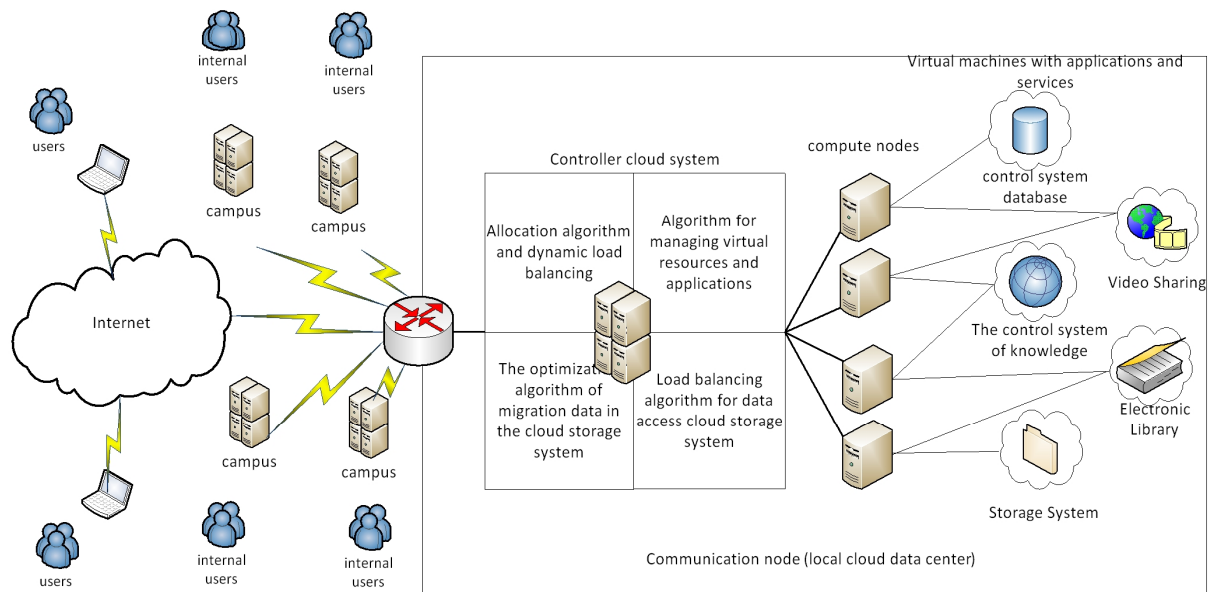


Fig. 1. Scheme of optimizing access to information system based on cloud computing

Our approach allows to consider the physical limitations of computing resources and organize the work of a cloud information system adjusting the number of instances of running applications based on the incoming flow of user requests.

V. EXPERIMENTAL PART

We have studied the work of the cloud information system with different parameters to evaluate the effectiveness of our virtual resource control algorithm. We have used the standard algorithms from the cloud system OpenStack [5] as reference for comparison in the experiment.

TABLE 1. Service efficiency of user requests

Systems	testing system	electronic library	video portal	testing system	electronic library	video portal
Experiment	1			3		
Number of requests	8000	1000	1000	1000	1000	8000
Volume of information	32650	9330	10340	4750	8210	92300
Number of serviced requests (without load balancing)	5443 (4352)	622 (418)	517 (356)	592 (465)	643 (512)	4320 (3985)
The intensity of service	90,71 (72,53)	10,36 (6,96)	8,61 (5,93)	9,8 (7,75)	10,71 (8,5)	72 (66,4)
Experiment	2			4	5	6
Number of requests	1000	8000	1000	10000	10000	10000
Volume of information	4250	67200	10670	41700	87600	108000
Number of serviced requests (without load balancing)	632 (525)	5384 (4625)	560 (376)	6753 (5642)	6351 (5215)	5860 (4129)
The intensity of service	10,5 (4,2)	89,73 (77,08)	9,3 (6,26)	112,5 (94,03)	105,85 (89,91)	97,6 (68,81)

In the experiment, we used the flow of requests similar to the real flow within the information system of distance learning. The number of concurrent requests received by the system was about 10,000, which is equal to the maximum number of potential users of the system.

All the user requests are classified into six user groups corresponding to the types of user behavior. The requests from the first three user groups directed to the allocated application using other applications at the same time. The groups from 4 to 6 simulate the work of the application in the case of

computing resource shortage because of an excess number of concurrent requests.

The intensity of using the system components (video portal, testing system, and electronic library) and the amount of the requested data were assigned for each user group. Experiment lasted for one hour which corresponds to the longest period of peak load in the real system. Experimental results are presented in the Table 1.

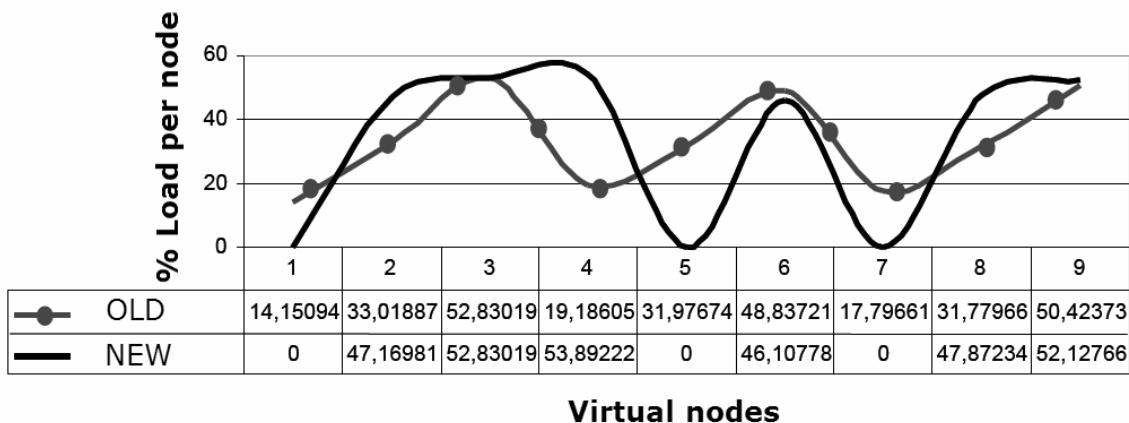


Fig. 2. Load balancing between nodes in the cloud system

The results of the experiments show a decrease of 12-15% of the number of service denials in accessing to multimedia services with limited resources. Within the experiment in the OpenStack cloud system we compared the consumption of virtual resources by the number of virtual servers for each of the subsystems.

Our control algorithm provides collaborative work of all running instances of applications in accordance with user requirements due to the optimal allocation of resources on each computing node. So the optimization algorithms may release 20 to 30% of the allocated resources (virtual servers) (Fig. 1).

VI. CONCLUSION

Thus, the effectiveness evaluation of the algorithm for control of virtual resources of the cloud system shows a performance boost from 12 to 15% compared to the standard. Our algorithm is very effective for high-intensity requests.

Besides the reduction of the number of allocated virtual resources allows to scale a cloud system more efficiently and provides a reserve for the case of increase in the intensity of using applications.

VII. REFERENCES

- [1] Qingjia Huang, Kai Shuang, Peng Xu, Jian Li, Xu Liu, Sen Su *Prediction-based Dynamic Resource Scheduling for Virtualized Cloud Systems* Journal of Networks, Vol 9, No 2 (2014), 375-383, Feb 2014. <http://doi:10.4304/jnw.9.2.375-383>
- [2] S. J. E. C. I. C. Clark, K. Fraser and A. Warfield, "Live migration of virtual machines," In Proc. NSDI, 2005.
- [3] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," in Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on. IEEE, 2007, pp. 119–128.
- [4] Q. Huang, S. Su, S. Xu, J. Li, P. Xu, and K. Shuang, "Migration-based elastic consolidation scheduling in cloud data center," in Proceedings of IEEE ICDCSW 2013.
- [5] *A scalable infrastructure for CMS data analysis based on OpenStack Cloud and Gluster file system* S Toor et al 2014 J. Phys.: Conf. Ser. 513 062047
- [6] A. Corradi, M. Fanelli, and L. Foschini. *VM Consolidation: a Real Case Based on OpenStack Cloud*. Future Generation Computer Systems, In Press.