SYRCoSE 2017

Editors:

Alexander S. Kamkin, Alexander K. Petrenko, and
Andrey N. Terekhov

Preliminary Proceedings of the 11th Spring/Summer Young Researchers' Colloquium on Software Engineering

Innopolis, June 5-7, 2017

2017

Preliminary Proceedings of the 11th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2017), June 5-7, 2017 – Innopolis, Republic of Tatarstan, Russian Federation.

The issue contains papers accepted for presentation at the 11th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2017) held in Innopolis, Republic of Tatarstan, Russian Federation on June 5-7, 2017.

The colloquium's topics include software specification and modelling, testing and verification, safety and security, operating systems, analysis of texts and social 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.

The event is sponsored by Russian Foundation for Basic Research (Project №17-07-20191).

Contents

Foreword ····· 5

Committees ····· 6

A Contract-Based Method to Specify Stimulus-Response Requirements
A. Naumchev, M. Mazzara, B. Meyer, J.-M. Bruel, F. Galinier, S. Ebersold ····· 8

Automated Type Contracts Generation in Ruby
N. Viuginov, V. Fondaratov ····· 14

Using Interface Patterns for Compositional Discovery of Distributed System Models
R. Nesterov, I. Lomazova ····· 17

An Online Tool for Requirements Engineering, Modeling and Verification of Distributed Software Based on MDD Approach
D. Lozhkina, S. Staroletov ····· 23

On Minimization of Timed Finite State Machines
A. Tvardovskii, N. Yevtushenko ····· 29

Experiments on Parallel Composition of Timed Finite State Machines
A. Sotnikov, N. Shabaldina, M. Gromov ····· 36

Mining Hybrid UML Models from Event Logs of SOA Systems
K. Davydova, S. Shershakov ····· 42

Tool for Behavioral Analysis of Well-Structured Transition Systems
L. Dworzanski, V. Mikhaylov ····· 51

Stochastic Methods for Analysis of Complex Hardware-Software Systems
A. Karnov, S. Zelenov ····· 57

Problems of Creation and Dynamic Analysis of Heterogeneous Models of Hardware/Software Systems
K. Agaeva ····· 62

Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions
A. Volkov, M. Mandrykin ····· 69

Static Verification of Configurations of Linux Operating System Kernel
S. Kozin, V. Mutilin ····· 74

A Technique for Parameterized Verification of Cache Coherence Protocols
V. Burenkov ····· 80

High-Level Model-Based Test Generation for Digital Hardware
M. Chupilko, A. Kamkin, M. Lebedev, S. Smolov ····· 87

Verification of 10 Gigabit Ethernet Controllers
M. Petrochenkov, R. Mushtakov, I. Stotland ····· 91

Creating Test Data for Market Surveillance Systems with Embedded Machine Learning Algorithms
O. Moskaleva, A. Gromova ····· 96

Initial Steps Towards Assessing the Usability of a Verification Tool
M. Khazeev, V. Rivera, M. Mazzara ··· 101

The Study into Cross-Site Request Forgery Attacks within the Framework of Analysis of Software Vulnerabilities
A. Barabanov, A. Lavrov, A. Markov, I. Polotnyanschikov, V. Tsirlov ··· 105

Dataflow Analysis for the Search of the Code Security Defects
S. Borzykh, A. Markov, A. Fadin, V. Tsirlov, P. Gusev ··· 110

Debugger for Real-Time OS: Challenges of Multiplatform Support
A. Emelenko, K. Mallachiev, N. Pakulin ··· 116

Using Modularization in Embedded OS
K. Mallachiev, N. Pakulin, A. Khoroshilov, D. Buzdalov ··· 120

Detecting and Tracking Near Duplicates in Software Documentation
D. Luciv ··· 125

Discovering Near Duplicate Text in Software Documentation
L. Kanteev, Yu. Kostyukov, D. Luciv ··· 130

The Program for Public Mood Monitoring through Twitter Content in Russian
S. Smetanin ··· 134

Narrabat – Prototype of News Retelling Service
I. Dolgaleva, I. Gorshkov, R. Yavorskiy ··· 138

The Program of Syntax-Based Sentiment Analysis
S. Smetanin ··· 142

Fast L^1 Gauss 2D Image Transforms
D. Bashkirova, S. Yoshizawa, R. Latypov, H. Yokota ··· 145

Real-time Video Stabilization using MEMS-sensors
A. Kornilova, Ia. Kirilenko, N. Zabelina ··· 150

Type-2 Fuzzy Rule-Based Model of Urban Metro Positioning Service
A. Gimaletdinova, K. Degtiarev ··· 156

A Modified Scrum Story Points Estimation Method Based on Fuzzy Logic Approach
S. Semenkovich, O. Kolekonova, K. Degtiarev ··· 164

The Mixed Chinese Postman Problem
M. Gordenko, S. Avdoshin ··· 172

Pareto-optimal Algorithms for Metric TSP
E. Beresneva (Chirkova), S. Avdoshin ··· 177

Rapid Prototyping of Smart Mirror for Smart Home Environment
S. Smetanin ··· 182

Inverse Kinematics in Ultralight UAV Control Problem with Additional On-Board Microcomputer
V. Kaliteevskiy, K. Amelin ··· 185

Foreword

Dear participants,

It is our pleasure to meet you at the 11th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE). This year's colloquium is hosted by Innopolis University, a young and very ambitious institution focused on education and research in the field of IT and robotics. The event is organized by Institute for System Programming of the Russian Academy of Sciences (ISP RAS), Saint-Petersburg State University (SPbSU), and Innopolis University.

SYRCoSE 2017's Program Committee (consisting of more than 50 members from more than 25 organizations) has selected 34 papers. Each submitted paper has been reviewed independently by two-three referees. The authors and speakers represent well-known universities, research institutes and companies including Bauman Moscow State Technical University, Exactpro Systems, Higher School of Economics, Innopolis University, ISP RAS, JetBrains, Kazan Federal University, Lomonosov Moscow State University, MCST, Moscow Institute of Physics and Technology, NPO Echelon, Polzunov Altai State Technical University, RIKEN, Southern Federal University, SPbSU, Tomsk State University, Toulouse University, and VERIMAG Laboratory (3 countries, 11 cities, and 18 organizations).

We would like to thank all of the participants of SYRCoSE 2017 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, Manuel Mazzara (Innopolis University), Andrey Belevantsev & Valery Ignatyev (ISP RAS), and Susanne Graf (VERIMAG Laboratory). We would also like to thank our sponsors: Russian Foundation for Basic Research (grant 17-07-20191), Federal Agency of Scientific Organizations, and Exactpro Systems. Our special thanks go to the local organizers, Adil Adelshin, Inna Baskakova, Manuel Mazzara, Bertrand Meyer, Nikolay Shilov, and Alberto Sillitti, for their invaluable help in organizing the colloquium in Innopolis.

Sincerely yours,

Alexander S. Kamkin
Alexander K. Petrenko
Andrey N. Terekhov

May 2017

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

Susanne Graf – France
VERIMAG Laboratory

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

Andrei V. Klimov – Russia
Keldysh Institute of Applied Mathematics 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

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

Victor M. Malyshko – Russia
Moscow State University

Tiziana Margaria – Ireland
Lero – The Irish Software Research Centre

Manuel Mazzara – Russia
Innopolis University

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

Alexander S. Mikhaylov – Russia
RN-Inform

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

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
A.P. Ershov Institute of Informatics Systems of RAS

Alberto Sillitti – Russia
Innopolis 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

Andrei N. Tiugashev – Russia
Samara State Aerospace University

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

Referees

A contract-based method to specify stimulus-response requirements

Alexandr Naumchev, Manuel Mazzara, Bertrand Meyer
Innopolis University
Innopolis, Russian Federation
{a.naumchev, m.mazzara, b.meyer}@innopolis.ru
Jean-Michel Bruel, Florian Galinier, Sophie Ebersold
Toulouse University
Toulouse, France
{bruel, galinier, ebersold}@irit.fr

***Abstract*—A number of formal methods exist for capturing stimulus-response requirements in a declarative form. Someone yet needs to translate the resulting declarative statements into imperative programs. The present article describes a method for specification and verification of stimulus-response requirements in the form of imperative program routines with conditionals and assertions. A program prover then checks a candidate program directly against the stated requirements. The article illustrates the approach by applying it to an ASM model of the Landing Gear System, a widely used realistic example proposed for evaluating specification and verification techniques.**

***Keywords*—Seamless Requirements, Design by Contract, AutoProof, Eiffel, Landing Gear System**

I. Overview and Main Results

The present article describes a technique for specification and verification of stimulus-response requirements using a general-purpose programming language (Eiffel) and a program prover (AutoProof [1]) based on the principles of Design by Contract [2].

Real-time, or reactive, systems are often run by a software controller that repeatedly executes one and the same routine and it is specified to take actions at specific time intervals or according to external stimuli [3]. This architecture is reasonable when the software has to react timely to non-deterministic changes in the environment. In this case the program should react to the external stimuli in small steps, so that in the event of a new change it responds timely.

Computation tree logics (CTL) [4] represent a frequent choice when it comes to capturing stimulus-response requirements. Although it may be easier to reason about requirements using declarative logic like CTL, the reasoning may be of little value for the software developer who will implement the requirements. Mainstream programming languages are all imperative, and the translation between declarative requirements and imperative programs is semi-formal.

Requirements have to be of imperative nature from the beginning. This would bridge the gap in how customers and developers understand them. For a software developer it is preferable to reason about the future program without switching to an additional formalism, notation and tools not connected to the original programming language and the IDE. The present article describes a technique to achieve this goal, in particular:

- Introduces the Landing Gear System (LGS) case study and the LGS baseline requirements (Section II).
- Generalizes the LGS baseline requirements, maps them to a well-established taxonomy, and complements the taxonomy (Section III).
- Provides a general scheme for capturing semantics of the stimulus-response requirements in the form of imperative program routines with assertions (Section IV).
- Exercises utility of the approach by applying it to an Abstract State Machine (ASM) specification of the Landing Gear System case study (Section V).
- Concludes the possibility of statically checking a sequential imperative program directly against a stimulus-response requirement whose semantics is expressed in the same programming language through conditionals, loops, and assertions (Section VII).

Application of the technique leads to discovery of an error in the published model of the LGS ASM [5]. The error is not present in the specification the authors have actually used for proving the properties, but the error has found its way into the publication.

II. The Landing Gear System

Landing Gear System was proposed as a benchmark for techniques and tools dedicated to the verification of behavioral properties of systems [6]. It physically consists of the landing set, a gear box that stores the gear in the retracted position, and a door attached to the box (Figure 1). The door and the gear are actuated independently by a digital controller. The controller reacts to changes in position of a handle in the cockpit by initiating either gear extension or retraction process. The task is to program the controller so that it correctly aligns in time the events of changing the handle's position and sending commands to the door and the gear actuators.

III. Stimulus-Response requirements

The LGS case study defines a number of requirements, including several for the normal mode of operation (Figure 2).

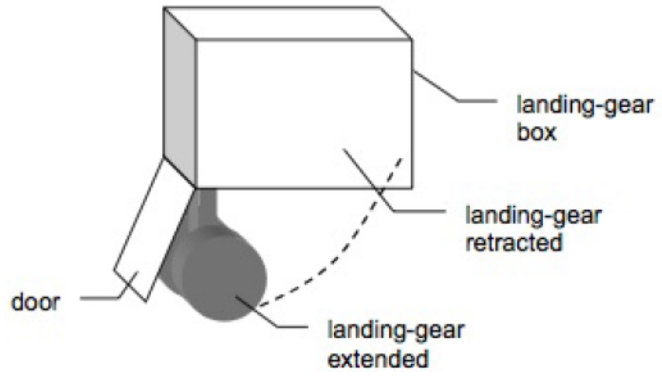


Fig. 1. Landing set (source: [6]).

($R_{11}bis$) When the command line is working (normal mode), if the landing gear command handle has been pushed DOWN and stays DOWN, then eventually the gears will be locked down and the doors will be seen closed.
($R_{12}bis$) When the command line is working (normal mode), if the landing gear command handle has been pushed UP and stays UP, then eventually the gears will be locked retracted and the doors will be seen closed.
(R_{21}) When the command line is working (normal mode), if the landing gear command handle remains in the DOWN position, then retraction sequence is not observed.
(R_{22}) When the command line is working (normal mode), if the landing gear command handle remains in the UP position, then outgoing sequence is not observed.

Fig. 2. Baseline LGS requirements.

The requirements communicate a common meaning of the form:

- If *stimulus* holds, then *response* will eventually hold in the future.

For requirement $R_{11}bis$,

$$stimulus \Leftrightarrow \text{“}The\ operation\ mode\ is\ normal\ and\ the\ handle\ is\ DOWN\text{”}$$

and

$$response \Leftrightarrow (stimulus \implies \text{“}The\ gears\ are\ down\ and\ the\ doors\ are\ closed\text{”})$$

The implication in the definition of $response$ reflects the “and stays DOWN” part of the original requirement.

In addition to that, requirements R_{21} and R_{22} communicate something else:

- Once *response* holds in the presence of *stimulus*, and *stimulus* holds forever, *response* will hold forever.

($R_{11}rs$) If the gears are locked extended and the doors are closed when the landing gear command handle is DOWN, this state will still hold if the handle stays DOWN.
($R_{12}rs$) If the gears are locked retracted and the doors are closed when the landing gear command handle is UP, this state will still hold if the handle stays UP.

Fig. 3. LGS response stability requirements.

A. Temporal interpretation of the requirements

The authors of the LGS ASM specification start with a ground model that satisfies a subset of requirements, and then refine the model to satisfy more requirements. The present article focuses on their ground model and the corresponding baseline requirements it covers (Figure 2). The work expresses the baseline requirements as CTL properties. The CTL interpretation assigns precise meanings to the requirements by assuming small-step execution semantics of ASM's. In particular, for requirements $R_{11}bis$ and $R_{12}bis$ “the future” means “after a finite number of execution steps”, while for R_{21} and R_{22} “the future” means “after one execution step”.

The finite number of steps in $R_{11}bis$ and $R_{12}bis$ may be unacceptably large though for a system like an LGS of an aircraft. In particular, flights have some expected durations, and the gears have to react to commands in some limited time frame as well. The following two major categories of stimulus-response requirements stem from the speculations above:

- *If stimulus holds, then response will hold in not more than k execution steps.*
  Requirements of this form are also called **maximal distance** requirements [7].
- *If stimulus holds, then response will hold in exactly k execution steps.*
  Requirements of this form are also called **exact distance**, or *delay* requirements.

These two categories are not enough though for capturing stimulus-response requirements. For example, if according to $R_{11}bis$ the gears are locked down and the doors seen closed as the result of the handle staying down, we want this state to be stable if the handle stays down. This leads us to stimulus-response requirements of the following form:

- *If response holds under stimulus, it will still hold after one execution step in the presence of that stimulus.*
  Let us call such requirements **response stability** requirements.

It makes sense to complement requirements ($R_{11}bis$) and ($R_{12}bis$) with the corresponding response stability requirements (Figure 3): not only do we want the LGS to respond to a change in the handle's position, but we also want it to maintain the response if the position does not change.

IV. Translation of Stimulus-Response Requirements

Assuming the presence of an infinite loop `from until False loop main end` that runs a

```
response_holds_within_k_steps
-- If stimulus holds,
-- response will hold within k steps.
  local
    steps: NATURAL
  do
    if (stimulus) then
      from
        steps := 0
      until
        response or (steps = k)
      loop
        main
        steps := steps + 1
      end
      check
        response
      end
    end
  end
```

Fig. 4. Representation of a maximal distance requirement. Regardless of the actual reason for the loop to terminate, the response has to hold if the stimulus held at the entry to the loop.

```
response_holds_in_k_steps
-- If stimulus holds,
-- response will hold in k steps.
  local
    steps: NATURAL
  do
    if (stimulus) then
      from
        steps := 0
      until
        response or (steps = k)
      loop
        main
        steps := steps + 1
      end
      check
        (response and (steps = k))
      end
    end
  end
```

Fig. 5. Representation of an exact distance requirement. Both of the loop exit conditions have to hold for the first time simultaneously if the stimulus held at the entry to the loop.

reactive system, a temporal stimulus-response requirement (Section III-A) takes the form of a routine with an assertion (**check end** construct in Eiffel). The authors draw this idea from the notion of a specification driver [8] - a contracted routine that forms a proof obligation in Hoare logic. AutoProof is a prover of Eiffel programs that makes it possible to statically check the assertions.

A. *Maximal distance*

In the representation of a maximal distance requirement (Figure 4) the "**if** `stimulus` **then**" clause captures the presence of the stimulus before the up-to-k-length execution fragment, and the "**check** `response` **end**" assertion expresses the need for the response upon completion of the sub-execution. The sub-execution may complete for two possible reasons: either occurrence of the response or consumption of all of the available k steps. In the both cases the response has to hold.

B. *Exact distance*

Representation of an exact distance requirement (Figure 5) is very similar to that one of a maximal distance, with the "**check** (`response` **and** (`steps` = `k`))**end**" assertion that makes the difference. Regardless of whether the loop terminates because of `response` or `steps`= `k`, the both have to hold upon the termination.

C. *Response stability*

Representation of a response stability requirement (Figure 6) says: whenever response holds under stimulus in a state, it will still hold in the presence of the same stimulus in the next state.

```
response_is_stable_under_stimulus
-- response keeps holding under stimulus.
  do
    if (stimulus and response) then
      main
      check
        (stimulus implies response)
      end
    end
  end
```

Fig. 6. Representation of a response stability requirement. If response holds under stimulus in some state, the response should hold in the next state in the presence of the same stimulus.

V. Applying the Translation Scheme to the Landing Gear Example

The article exercises the approach on the LGS ASM specification, which is operational by the definition and thus is a subject for translation into an imperative program. For this reason the present section starts with explanation of the rules according to which the authors converted the original specification into an Eiffel program.

A. *Translation of ASM specifications*

An ASM specification is a collection of rules taking one of the following three forms [9]: assignment (Section V-A1), do-in-parallel (Section V-A2), and conditional (Section V-A3). If we have general rules for translating these operators into Eiffel then we will be able to translate an arbitrary ASM into an Eiffel program.

1) *Assignment:*

An ASM assignment looks as follows:

$$f(t_1, .., t_j) := t_0 \qquad (1)$$

The semantics is: update the current content of location $\lambda = (f, (a_1, .., a_j))$, where a_i are values referenced by t_i, with the value referenced by t_0.

In Eiffel locations are represented with class attributes, so an ASM's location update corresponds in Eiffel to an attribute assignment.

2) Do-in-parallel: An ASM can apply several rules simultaneously in one step:

$$R_1||...||R_k \tag{2}$$

In order to emulate a parallel assignment in a synchronous setting, one needs to assign first to fresh variables and then assign their values to the original ones. For example, an ASM do-in-parallel statement

$$a, b := max(a - b, b), min(a - b, b) \tag{3}$$

in Eiffel would look like

```
local
  a_intermediate, b_intermediate: INTEGER
do
  a_intermediate := max (a−b, b)
  b_intermediate := min (a−b, b)
  a := a_intermediate
  b := b_intermediate
end
```

An attempt to update in parallel identical locations in an ASM corresponds semantically to a crash. The translation scheme not only preserves but strengthens this semantics: an Eiffel program with two local variables declared with identical names will not compile.

3) Conditional: An ASM conditional **if** t **then** R_1 **else** R_2 carries the same meaning as in Eiffel, so the translation is straightforward.

B. Ground model

Translation of the original LGS ASM specification into Eiffel is publicly available in a GitHub repository [10] and needs clarification too.

The baseline LGS requirements (Figure 2) talk about normal mode of operation. The ground ASM specification captures the normal mode through a model invariant, while the Eiffel translation introduces a special boolean query `is_normal_mode` for this purpose. The reason for that is rather technical and has to do with the current limitations in the underlying verification technology. The translation also contains a number of annotations for disabling the complications of the underlying verification methodology [11]. Special comments highlight the annotations and tell explicitly that they have nothing to do with the problem at hand.

The repository contains two versions of the ground model, `GROUND_MODEL_ORIGINAL` and `GROUND_MODEL`. The original one keeps the error from the ASM model, which is not handling opening doors case in the extension sequence. The second version contains the translation without the error.

C. Requirements

The two classes include the translations of the baseline requirements plus the response stability requirements introduced in the present article. We do not discuss all of them here: requirements $(R_{11}bis)$ and $(R_{12}bis)$, (R_{21}) and (R_{22}), $(R_{11}rs)$ and $(R_{12}rs)$ are pairwise similar, which is why we prefer to pick one from each pair.

Translation of requirement `r11_bis` (Figure 7) is an application of the `response_holds_within_k_steps` pattern (Figure 4), where:

- `stimulus` equates to:

```
is_normal_mode and
  (handle_status= is_handle_down)
```

- `response` equates to:

```
(not (is_normal_mode and
  (handle_status= is_handle_down))) or
((gear_status= is_gear_extended) and
  (door_status= is_door_closed))
```

The idea behind the response is that there may be two reasons for the gear not to extend and the door not to close:

- An abnormal situation that leads to quitting the normal mode.
- The crew changes their mind and pushes the handle up.

VI. Related work

Modeling of real-time computation and related requirements is a well-investigated matter [12]. Representation of real-time requirements, expressed in general or specific form, is a challenging task that has been attacked by the use of several formalisms both in sequential and concurrent settings, and in a broad set of application domains. The difficulty (or impossibility) to fully represents general real-time requirements other than in natural language, or making use of excessively complicated formalisms (unsuitable for software developers), has been recognized.

In [13] the domain of real-time reconfiguration of system is discussed, emphasizing the necessity of adequate formalisms. The problem of modeling real time in the context of services orchestration in Business Process, and in presence of abnormal behavior has been examined in [14] and [15] by means, respectively, of process algebra and temporal logic. Modeling of protocols also requires real-time aspects to be represented [16]. Event-B has also been used as a vector for real-time extension [17] in order to handle embedded systems requirements.

In all these studies, the necessity emerged of focusing on specific typology of requirements using ad-hoc formalisms and techniques, and making use of abstractions. The notion of "real-time" is often abstracted as *number of steps*, a metric commonly used. In this paper we follow the same approach, inheriting both strength (simplicity of the model and effectiveness for applicative purposes) and limitations (temporal logic and time automata themselves miss to capture a precise notion of *real-time*).

VII. Conclusions and future work

Software developers reason in an *imperative/operational* manner. This claim is supported both by anecdotal experience and by empirical evidence [18]. Requirements expressed in imperative/operational fashion would therefore results of easier comprehensions for developers and would simplify the process of negotiation behind requirements elicitation.

```
r11_bis
-- If (is_normal_mode and (handle_status = is_handle_down)) hold and remain,
-- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) will hold within 10 steps.
 local
   steps: NATURAL
 do
   if (is_normal_mode and (handle_status = is_handle_down)) then
     from
       steps := 0
     until
       (not (is_normal_mode and (handle_status = is_handle_down))) or
       ((gear_status = is_gear_extended) and (door_status = is_door_closed)) or
       (steps= 10)
     loop
       main
       steps := steps + 1
     end
     check
       (not (is_normal_mode and (handle_status = is_handle_down))) or
       ((gear_status = is_gear_extended) and (door_status = is_door_closed))
     end
   end
 end
```

Fig. 7. Translation of the “r11_bis” requirement.

```
r21
-- If (is_normal_mode and (handle_status = is_handle_up)) holds and remains,
-- (gear_status ≠ is_gear_extending) will hold within 1 step.
 local
   steps: NATURAL
 do
   if (is_normal_mode and (handle_status = is_handle_up)) then
     from
       steps := 0
     until
       (not (is_normal_mode and (handle_status = is_handle_up))) or
       (gear_status ≠ is_gear_extending) or
       (steps = 1)
     loop
       main
       steps := steps + 1
     end
     check
       (not (is_normal_mode and (handle_status = is_handle_up))) or
       (gear_status ≠ is_gear_extending)
     end
   end
 end
```

Fig. 8. Translation of the “r21” requirement.

```
 r11_rs
 -- ((gear_status = is_gear_extended) and (door_status = is_door_closed)) keeps holding under
 -- (is_normal_mode and (handle_status = is_handle_down))
   do
     if ((is_normal_mode and (handle_status = is_handle_down)) and
        ((gear_status = is_gear_extended) and (door_status = is_door_closed))) then
       main
       check
         ((is_normal_mode and (handle_status = is_handle_down)) implies
           ((gear_status = is_gear_extended) and (door_status = is_door_closed)))
       end
     end
   end
```

Fig. 9. Translation of the “r11_rs” requirement.

In the method described in this paper, requirements are expressed in a formalism (or language) that seamlessly stay the same along the whole process, without the need of switching between different instruments or mental paradigms. At the same time, the linguistic tool used to define them also allows for automatic verification of correctness.

The meaning of correctness here remains subject to the assumption that requirements engineers and stakeholders agree on a list of desiderata that is indeed the intended one. Assuming a non-faulty process of intention transferring (and this assumption is common to any other approach too), requirements are now more easily manageable by software engineerings all the way from elicitation to verification.

The result of elicitation process is a set of requirements in natural language. The full realization of the presented method would imply an automatic (or semi-automatic) translation from natural language into a structured representation that, although completely intuitive for software developers, it is possibly not easy to manage for average stakeholders. The first part of this process, i.e., the translation from natural language into the current representation (and back) is under development. A tool automatically translates semi-structured natural language into the Hoare-triple-based representation [19], allowing also the opposite direction, i.e. back to natural language [20], so that software engineers would be able to negotiate back requirements with stakeholders using a format they would comprehend. The role of the requirement engineers would then consist in concluding the elicitation phase with a set of requirements in semi-structured natural language, which the tool would be able to process in an entirely automatic manner.

This paper supports the idea of seamless development describing a method supported by a formalism that stay the same along the whole process, from requirements to deployment. Alternative approaches have also been experimented which make use of formalism-based toolkits, where ad hoc notations are adopted for each development phase [21].

References

[1] J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, "Autoproof: Auto-active functional verification of object-oriented programs," *arXiv preprint arXiv:1501.03063*, 2015.

[2] B. Meyer, *Touch of Class: learning to program well with objects and contracts*. Springer, 2009.

[3] I. J. Hayes, M. A. Jackson, and C. B. Jones, *Determining the Specification of a Control System from That of Its Environment*, pp. 154–169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[4] E. Clarke and E. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," *Logics of programs*, pp. 52–71, 1982.

[5] P. Arcaini, A. Gargantini, and E. Riccobene, "Modeling and analyzing using asms: the landing gear system case study," in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 36–51, Springer, 2014.

[6] F. Boniol and V. Wiels, "The landing gear system case study," in *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pp. 1–18, Springer, 2014.

[7] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[8] A. Naumchev and B. Meyer, "Complete contracts through specification drivers," in *2016 10th International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 160–167, July 2016.

[9] Y. Gurevich, "Sequential abstract-state machines capture sequential algorithms," *ACM Transactions on Computational Logic (TOCL)*, vol. 1, no. 1, pp. 77–111, 2000.

[10] A. Naumchev, "Lgs asm ground model in eiffel.." https://github.com/anaumchev/lgs_ground_model, 2017.

[11] N. Polikarpova, J. Tschannen, C. A. Furia, and B. Meyer, "Flexible invariants through semantic collaboration," in *FM 2014: Formal Methods*, pp. 514–530, Springer, 2014.

[12] H. Yamada, "Real-time computation and recursive functions not real-time computable," *IRE Transactions on Electronic Computers*, vol. EC-11, pp. 753–760, Dec 1962.

[13] M. Mazzara and A. Bhattacharyya, "On modelling and analysis of dynamic reconfiguration of dependable real-time systems," in *Proceedings of the 2010 Third International Conference on Dependability*, DEPEND '10, (Washington, DC, USA), pp. 173–181, IEEE Computer Society, 2010.

[14] M. Mazzara, "Timing issues in web services composition," in *Formal Techniques for Computer Systems and Business Processes, European Performance Engineering Workshop, EPEW 2005 and International Workshop on Web Services and Formal Methods, WS-FM 2005, Versailles, France, September 1-3, 2005, Proceedings*, pp. 287–302, 2005.

[15] L. Ferrucci, M. M. Bersani, and M. Mazzara, "An LTL semantics of businessworkflows with recovery," in *ICSOFT-PT 2014 - Proceedings of the 9th International Conference on Software Paradigm Trends, Vienna, Austria, 29-31 August, 2014*, pp. 29–40, 2014.

[16] M. Berger and K. Honda, "The two-phase commitment protocol in an extended pi-calculus," *Electr. Notes Theor. Comput. Sci.*, vol. 39, no. 1, pp. 21–46, 2000.

[17] A. Iliasov, A. Romanovsky, L. Laibinis, E. Troubitsyna, and T. Latvala, "Augmenting event-b modelling with real-time verification," in *Proceedings of the First International Workshop on Formal Methods in Software Engineering: Rigorous and Agile Approaches*, FormSERA '12, 2012.

[18] D. Fahland, D. Lübke, J. Mendling, H. Reijers, B. Weber, M. Weidlich, and S. Zugal, *Declarative versus Imperative Process Modeling Languages: The Issue of Understandability*. Springer Berlin Heidelberg, 2009.

[19] A. Bormotova, "Translation of natural language into hoare triples." https://github.com/An-Dole/Semantic-mapping.

[20] V. Skukov, "Translation of hoare triples into natural language." https://github.com/flosca/hybrid.

[21] R. Gmehlich, K. Grau, F. Loesch, A. Iliasov, M. Jackson, and M. Mazzara, "Towards a formalism-based toolkit for automotive applications," in *1st FME Workshop on Formal Methods in Software Engineering, FormaliSE 2013, San Francisco, CA, USA, May 25, 2013*, pp. 36–42, 2013.

Automated Type Contracts Generation in Ruby

Nickolay Viuginov
Software engineer chairs
St.-Petersburg State University
St.-Petersburg, Russia
Email: viuginov.nickolay@gmail.com

Valentin Fondaratov
Team Lead in RubyMine
JetBrains
St.-Petersburg, Russia
Email: fondarat@gmail.com

***Abstract*—Elegant syntax of Ruby language pays back when it comes to finding bugs in large codebases. Static analysis is hindered[1] by specific capabilities of Ruby, such as defining methods dynamically and evaluating string expressions. Even in dynamically typed languages, type information is very useful, because of better type safety and more reliable checks. One may annotate the code with YARD (Ruby documentation tool) which also enables improved tooling such as code completion.**

This paper reports a new approach to type annotations generation. We trace direct method calls while the program is running, evaluate types of input and output variables and use this information to derive implicit type annotations.

Each method or function is associated with a finite-state automaton, to which all variants of typed signatures are added. Then an effective compression technique is applied to the automaton, which reduces the cost of storage and allows to display the collected information in a human-readable form.

***Index Terms*—Ruby, Dynamically typed languages, Ruby VM, YARV, Method signature, Type inference, Static code analysis.**

I. Introduction

Developers suffer from time-consuming investigations with a goal to understand why the particular piece of code does not work as expected. The dynamic nature of Ruby allows for great possibilities which has a drawback — the codebase as a whole becomes entangled and investigations become more difficult compared to the statically typed languages like Java or C++. Another downside of its dynamic features is a drastic static analysis performance reduction due to inability to resolve some symbols reliably.

Consider the dynamic method creation which is often done with `define_method` call. Names and bodies of dynamically created methods may be calculated at runtime[2].

```ruby
class User
  ACTIVE = 0
  INACTIVE = 1
  PENDING = 2

  attr_accessor :status

  def self.states(*args)
    args.each do |arg|
      define_method "#{arg}?" do
        self.status == User.const_get(arg.upcase)
      end
    end
  end
  states :active, :inactive, :pending
end
```

One of the possible workarounds is using code documentation tools like RDoc or YARD. As `@param` and `@return` annotations may help, they have several drawbacks, too:

- the type system used for documenting attributes, parameters and return values is pretty decent however it is not clear how to connect the types with each other. For example, `def []=` for array usually returns the same type as the second arg taking any type so in YARD this will look like `@param value [Object]`, `@return [Object]` which is not really helpful.
- from some perspective, such documentation itself a kind of contradicts the purpose of Ruby — to be as short, natural and expressive as possible.

The proposed approach is inspired by the way people tackle this problem manually: one may run or debug the program to inspect some valuable info about the code they're interested in. This suggests that collecting direct input and output types of the method dispatches during the program execution, post-processing and structuring of this data will make up implicit type annotations. As the process is automated, one can retrieve a plenty of information about the covered code.

The collected information not only might be used for YARD annotations generation but also could be stored in a public database to be shared and reused by different users in order to maximize the coverage of the analyzed code and the quality of the results. Moreover, generated implicit annotations can be built into the static analysis tools[3]) to improve existing and provide additional checks and code completion suggestions.

The project implementation can be divided into two main parts:

- At the first stage, the information about called methods and their input and output types is being collected though-out the script execution. It is very important to collect the necessary information as quickly as possible not to force users to wait for script completion many times longer compared to the case of the regular execution. To achieve this we implement a native extension, which receives all the necessary information directly from the internal stack of the virtual machine instead of using the standard API provided by language.
- At the second stage, the data obtained in the first stage is structured, reduced to a finite-state automaton and prepared for further use in code insight. This storage

scheme was chosen because of the ability to quickly obtain a regular expression that is easily perceived by a human.

II. Related works

For Ruby, as for most dynamically typed languages, there are tools for source code analysis. But they do not allow to statically identify all errors associated with type mismatch. Here are some of them:

- Rubocop [4] - A Ruby static code analyzer, based on the community Ruby style guide. But it does not allow to detect actual errors in types
- Ruby-lint - A tool for detecting syntax errors, such as undeclared variables, an invalid argument set for calling a method, or unreachable sections of code.
- Diamondback Ruby [5] - an extension to Ruby that aims to bring the benefits of static typing to Ruby. But at the moment it's impossible to analyze even the standard Ruby library.

III. Collecting information about method calls

Method parameters in Ruby have the following structure:

```
def m(a1, a2, ..., aM,   # mandatory(req)
    b1=(...), ..., bN=(...), # optional(opt)
    *c,                  # rest
    d1, d2, ..., dO,     # post
    e1:(...), ..., eK:(...), # keyword
    **f,                 # keyword_rest
    &g)                  # block
```

TracePoint is a API allowing to hook several Ruby VM events like method calls and returns and get any data through Binding, which encapsulates the execution context(variables, methods) and retain this context for future use.

```
def foo(a, b = 1)
  b = '1'
end

TracePoint.trace(:call, :return) do |tp|
  binding = tp.binding
  method = tp.defined_class.method(tp.method_id)
  p method.parameters
  puts tp.event, (binding.local_variables.map do |v|
    "#{v}->#{binding.local_variable_get(v).inspect}"
  end.join ', ')
end

foo(2)
```

```
[[:req, :a], [:opt, :b]]
call
a->2, b->1
[[:req, :a], [:opt, :b]]
return
a->2, b->"1"
```

The big disadvantage of this approach is that calculation of full execution context is a time-consuming operation. But later we will need information only about a small part of it. Namely: types of arguments, types and names of method parameters. Creating a native extension for the Ruby VM[6], which will receive information about the method name directly from YARV instruction list (Figure 1) and receive information about argument types directly from the internal stack. Code

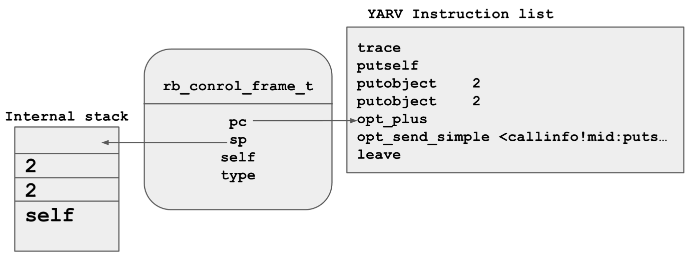


Fig. 1. YARV's internal registers.

analysis often handles direct method calls, so it is important to separate which arguments were directly passed to the method by the user, and which ones were assigned the default values. When Ruby VM hook the call event, all not specified optional arguments already initialized with default values. So we need to build one more native extension and gem this information from internal stack. Lets take a look at simple Ruby method with optional parameter and on appropriate bytecode.

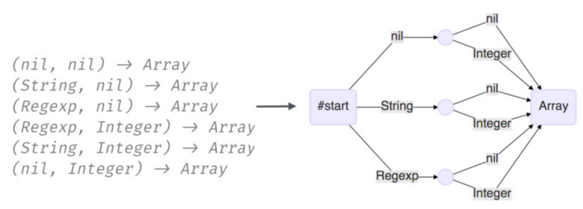
```
def foo(a, b=42, kw1: 1, kw2:, kw3: 3)
  #...
end

foo(1, kw1: '1', kw2: '2')
```

```
== disasm: #<ISeq:<compiled>@<compiled>>============
0000 trace      1
0002 putspecialobject 1
0004 putobject   :foo
0006 putiseq     foo
0008 opt_send_without_block
    <callinfo!mid:core#define_method, argc:2,
    ARGS_SIMPLE>
0011 pop
0012 trace      1
0014 putself
0015 putobject_OP_INT2FIX_O_1_C_
0016 putstring  "1"
0018 putstring  "2"
0020 opt_send_without_block <callinfo!mid:foo,
    argc:3, kw:[kw1,kw2], FCALL|KWARG>
0023 leave
== disasm: #<ISeq:foo@<compiled>>==================
0000 putobject   42
0002 setlocal_OP__WC__0 6
0004 trace      8
0006 putnil
0007 trace      16
0009 leave
```

So we need to find bytecode instruction for current method dispatch. For this, it is necessary to find caller control frame, and get last executed instruction in this frame. Thats how we get number of arguments(argc:) and list of key words(kw:[])

IV. Transforming raw call data into contracts

Huge amount of raw signatures received from the Ruby process must be structured and processed so that it can be easily used and perceived. Each traced method is associated with a finite-state automaton. This storage structure allows you to quickly add raw data obtained from the Ruby process. It is also can easily be reduced to a human-readable regular expression.



```
(nil, nil) → Array
(String, nil) → Array
(Regexp, nil) → Array
(Regexp, Integer) → Array
(String, Integer) → Array
(nil, Integer) → Array
```

Fig. 2. Example of generating a non-minimized automaton.

In each automaton there is a single starting vertex and a single terminal vertex. To the automaton consistently added words obtained by concatenating signatures and corresponding output types. Then the minimization algorithm[7] is applied to this automaton. Quite often there are situations when the types of the two or more arguments of the method always coincide. Or even the type of the result coincides with type of one of the arguments. Consider an algorithm of processing such methods using method `equals` as an example.

```ruby
def equals(a, b)
  raise StandardError if a.class != b.class
  a == b
end
p equals(1, 1) # (Integer, Integer) -> TrueClass
p equals(1, 2) # (Integer, Integer) -> FalseClass
...
```

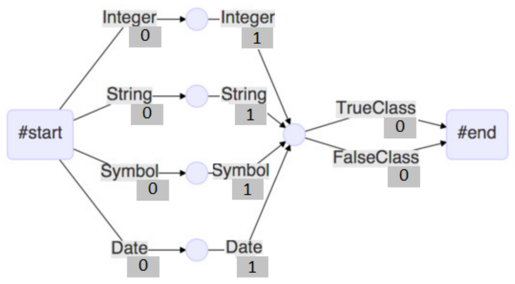


Fig. 3. Automaton with counted bit masks.

In this case, the types of arguments are the same, so the automaton can be markedly reduced. The algorithm consists in storing a bit mask on each automaton transition. Bit with one indicates a match of the type written on the edge with the corresponding previous type (Fig 3). In case the mask is greater than 0, the type written on the edge can be replaced with a mask. Subsequently, the transition along the edge will be carried out only when a readable signature is applied to the mask. After that, minimization algorithm is applied to the automaton one more time.

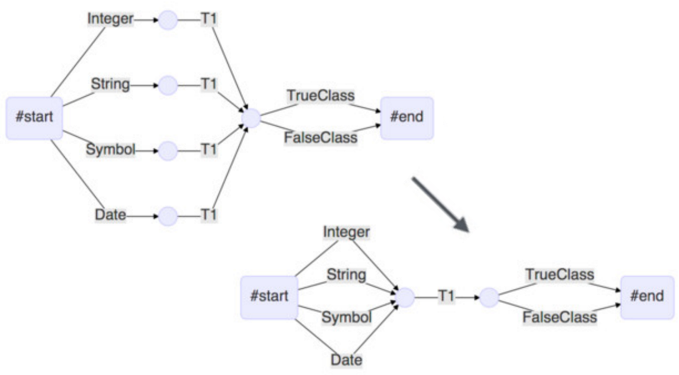


Fig. 4. Minimizing the automaton with reference edges.

When during the code analysis it will be necessary to calculate the type returned by the method, we simply read through the automaton the set of input types of this method, and all the transitions from the vertex in which we turn out to be the desired output types.

V. Conclusion

The paper describes the approach to generation of implicit type annotations.

This approach provides information about the types of methods that can not be obtained by static analysis of the source code in case if it is possible to understand in which library the method was declared and resolve the method receiver. This approach is useful for methods that are declared dynamically or in their body there are complex syntactic constructions, for example evaluating a string variable . In addition, this approach can be applied to other languages with dynamic typing, such as Python or JavaScript.

At the moment, we are working on optimizing the time of collecting of statistics. In the future, it will be necessary to test the approach, carry out load testing and implement the prototype in the existing static analysis of the Ruby language.

References

[1] Brianna M. Ren. The ruby type checker.
[2] blog.codeclimate. Gradual type checking for ruby, 2014.
[3] O. Shivers. *Control flow analysis in scheme*. ACM SIGPLAN 1988 conference on Programming language design and implementation, 1988.
[4] Bozhidar Batsov. Rubocop, 2017.
[5] Jeff Foster Mike Hicks Mike Furr, David An. Diamondback ruby guide, 2009.
[6] Pat Shaughnessy. *Ruby Under a Microscope*. No Starch Press, 2013.
[7] Jeffrey D. Ullman John E. Hopcroft, Rajeev Motwani. *Introduction to Automata Theory*. Addison-Wesley, 2001.

Using Interface Patterns for Compositional Discovery of Distributed System Models

Roman A. Nesterov*, Irina A. Lomazova†
National Research University Higher School of Economics
20 Myasnitskaya Ulitsa, 101000, Moscow, Russia
E-mail: *ranesterov@edu.hse.ru, †ilomazova@hse.ru

***Abstract*—Process mining offers various tools for studying process-aware information systems. They mainly involve several participants (or agents) managing and executing operations on the basis of process models. To reveal the actual behavior of agents, we can use process discovery. However, for large-scale processes, it does not yield models which help understand how agents interact since they are independent and their concurrent implementation can lead to a very sophisticated behavior. To overcome this problem, we propose interface patterns which allow getting models of multi-agent processes with a clearly identified agent behavior and interaction scheme as well. The correctness of patterns is provided via morphisms. We also conduct a preliminary experiment, results of which are highly competitive compared to the process discovery without interface patterns.**

***Index Terms*—Petri nets, interface patterns, synchronization, composition, morphisms, process discovery, multi-agent systems, distributed systems**

I. Introduction

Process mining is the relatively new direction in studying process-aware information systems. They include information systems managing and executing operational processes which involve people, applications and information resources through process models [1, p. 3]. Examples of these systems include workflow management systems, business process management systems, and enterprise information systems. The underlying interactions among participants (also called agents) of process-aware information systems are intrinsically distributed multi-agent systems. An agent acts autonomously, but it can interact with the others via shared resources, restrictions, and other means. Process mining helps to extract a model of this system for further study from a record of its implementation called an event log. However, extracted models are hard for analysis since there might be complex interactions among process participants the number of those can be significant.

In this paper, we propose a compositional approach to address this problem. Given an event log of a distributed system, we can filter it by agents and mine a model of each agent. Then, agent models can be composed to get a complete model of a multi-agent distributed system which might be simulated. Composing agent models allows us to obtain more structured models compared to models extracted from complete logs since the behavior of an agent can be clearly identified. We compose agent models via interface patterns which describe how they intercommunicate. This approach was presented at TMPA-2017 [2]. The formal proof of the composition correctness is based on using net morphisms [3]. Moreover, interface patterns allow inheriting deadlock-freeness and proper termination from agents by construction.

We conduct a preliminary experiment on using one interface pattern for mining multi-agent models. The outcomes are evaluated with the help of conformance checking quality dimensions [1, p. 128], [4] and complexity metrics [5].

This paper is structured as follows. The next section provides an overview of process discovery and compositional approaches. In Section 3 we introduce basic terms which are used in the paper. Section 4 shows a general description of the compositional approach to process discovery. Section 5 briefly introduces how we compose agent models using interface patterns and net morphisms. In Section 6 we describe the preliminary experiment and analyze results.

II. Related work

There exist three types of process mining, namely discovery, conformance, and enhancement. Process discovery produces a process model out of an event log – a record of implemented activities. Existing discovery approaches can yield a model in a variety of notations including Petri nets, heuristic nets, process trees, BPMN, and EPC. Petri nets are the most widespread process model representations discovered from event logs. Conformance checking is used to check whether a discovered model corresponds to an input event log and to identify probable deviations. The main idea of enhancement is to improve existing processes using knowledge of actual processes (usually denoted AS-IS) obtained from event logs.

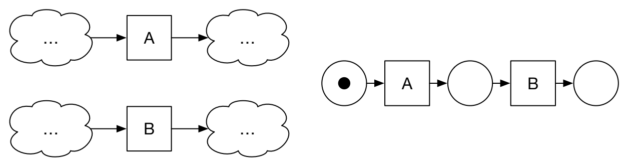
Process discovery offers several methods to be used for constructing models from event logs. One of the first and the most straightforward discovery approach is α**-algorithm** which identifies ordering relations among activities in logs, but it has severe usage limitations connected with cycles and the overall quality of obtained models [1, pp. 136-139]. It has several refined versions and improvements, for example [6], but there are other more sophisticated and efficient discovery algorithms. S. Leemans et al. [7] has proposed **inductive miner** allowing to extract process models from logs containing infrequent or incomplete behavior as well as dealing with activity lifecycle when there are separate actions of start and finish for each activity. Apart from that, inductive miner always produces well-structured models in the form of Petri nets. **HeuristicsMiner** is another process discovery algorithm proposed by A. Weijters et al. [8]. It can process event logs with a lot of noise (excessive activities) and also deals with infrequent process behavior. HeuristicsMiner uses intermediate casual matrices and produces heuristics net which can easily be converted into Petri nets and applied for other notations including EPC, BPMN, and UML. S. van Zelst et al. [9] proposed the approach to process discovery based on **integer linear programming and theory of regions**. Their algorithm can produce Petri nets with complex control flow patterns, and its recent improvements guarantee the structural correctness of discovered models. C. Gunther and W. van der Aalst have proposed **adaptive fuzzy mining** approach [10] to deal with unstructured processes extracted from event logs since they can produce different abstractions of processes distinguishing "important" behavior.

Since state-of-the-art process discovery algorithms can deal with complex process behavior, the other problem is to obtain models that are appropriate concerning their structure. There is a so-called continuum of processes ranging from highly structured processes (Lasagna models) to unstructured processes (Spaghetti models) [1, pp. 277-317]. The problem of obtaining well-structured models is extensively studied in the literature. Researchers offer different techniques to improve model structure [11], and to produce already well-structured process models [12], [13], [14]. In the case of multi-agent and distributed systems using well-structured models should also allow identifying agent behavior clearly for the model understandability improvement.

We suggest discovering models of agents independently and then composing them together to produce a structured multi-agent system model with the clearly visible behavior of each agent. Several compositional approaches for process discovery have been proposed. In [15] A. Kalenkova et al. have shown how to obtain a more readable model from an event log by decomposing extracted transition systems. A special tecnique to deal with cancellations in process implementation and to produce clear and structured process models which can contain cancellations have been studied in [16].

Correct coordination of system components is an error-prone task. Their interaction can generate complex behavior. The majority of process discovery tools produce Petri nets, and a large amount of literature has investigated the problem of composing Petri nets. They can be composed via straightforward merging of places and transitions [17, pp. 75-80], but the composition result will not preserve component properties. One of the possible ways to achieve inheritance of component behavioral properties is to use morphisms [18]. Special constructs for composing Petri net based on morphisms were studied in [3], [19], [20]. The key idea of this approach is that distributed system components refine an abstract interface describing the interactions between them. In [21] I. Lomazova has proposed a compositional approach for a flexible re-engineering of business process by using a system of interacting workflow nets. There also exists a several techniques for compositional synthesis of web services [22]. However, in [23] R. Hamadi and B. Benatallah have proposed an algebraic approach to the regular composition of services.

These compositional approaches do not let specify the explicit order of inner behavior of two interacting components. This situation is schematically represented in Fig. 1. Having two discovered component models with always executable actions **A** and **B**, we want to require that they interact in a way that **A** is implemented before **B**. This way of intercommunication is also shown in the form of Petri net.



(a) Modeling components (b) Interaction scheme

Fig. 1. Defining relations on inner actions of components

In [2] we have proposed a solution to this problem and two other patterns for composing two interacting components. The obtained composition inherits properties, such as deadlock-freeness and proper termination, from components.

In this paper we show how these patterns can be used for discovering a multi-agent system model from an event log in a compositional way. Applying compositional patterns allows us to obtain a more readable model improving time complexity due to the parallelization of process discovery.

We can assess process models obtained from event logs against four standard quality dimensions – fitness, precision, generalization, and simplicity [4]. **Fitness** identifies how accurately an extracted model can replay a source event log. **Precision** indicates a fraction of a behavior allowed by the model but not seen in the event log. **Generalization** tries to measure the extent to which the model will be able to implement the behavior of the process unseen so far in the log. **Simplicity** focuses on assessing structural complexity alongside with other graph characteristics - a number of elements and a structuredness measure [5].

III. Preliminary definitions

A. Petri nets

We use Petri nets [17] to represent agent models and an interaction scheme called interface.

Definition 1: A **multiset** m over a set S is a function $m\colon S \to \mathbb{N} \cup \{0\}$. Let m, m' be two multisets, $m' \subseteq m$ iff $\forall s \in S\colon m'(s) \leq m(s)$. Also, $\forall s \in S\colon (m + m')(s) = m(s) + m'(s)$ and $(m - m')(s) = max(0, m(s) - m'(s))$.

Then, an ordinary set is a multiset in which distinct elements occurr only once.

Definition 2: A **Petri net** is a bipartite graph $N = (P, T, F, m_0, L)$, where:

1) $P = \{p_1, p_2, ..., p_n\}$ – a finite non-empty set of places.
2) $T = \{t_1, t_2, ..., t_m\}$ – a finite non-empty set of transitions, $P \cap T = \varnothing$.
3) $F \subseteq (P \times T) \cup (T \times P)$ – a flow relation.

4) $m_0 \colon P \to \mathbb{N} \cup \{0\}$ – a multiset over P, initial marking.
5) $L \colon T \to \mathcal{A} \cup \{\tau\}$ – a labeling for transitions, where τ is a name for **silent** transitions.

Pictorially, places are shown as circles, and transitions are shown as boxes (silent transitions are depicted by black boxes). A flow relation is depicted by directed arcs (see Fig. 2).

Let $X = P \cup T$. We call a set ${}^\bullet x = \{y \in X | (y, x) \in F\}$ a **preset** of x and a set $x^\bullet = \{y \in X | (x, y) \in F\}$ – a **postset** of x. Also, ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$ is a **neighborhood** of x.

The behavior of Petri nets is defined by the firing rule, which specifies when an action can occur, and how it modifies the overall state of the system.

A marking $m \colon P \to \mathbb{N} \cup \{0\}$ **enables** a transition t, denoted $m[t\rangle$, if ${}^\bullet t \subseteq m$. The t firing in m leads to m', denoted $m[t\rangle m'$, where $m' = m - {}^\bullet t + t^\bullet$. When $\forall t \in T$ and $\forall w \in T^*$, $m[tw\rangle m' = m[t\rangle m''[w\rangle m'$, w is called the **firing sequence**. We denote a set of all firing sequences of a net N as $FS(N)$.

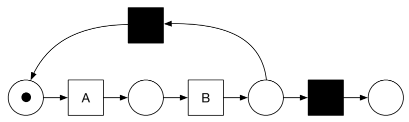


Fig. 2. A Petri net

We call a marking m **reachable** from m_0 if $\exists w \in FS(N) : m_0[w\rangle m$. A set of all markings reachable from m_0 is denoted by $[m_0\rangle$. So, $[m\rangle$ is a set of all markings reachable from m. A net N is **safe** iff $\forall p \in P, \forall m \in [m_0\rangle \colon m(p) \leq 1$.

A marking m_f is called **final** if $\forall p \in m_f \colon p^\bullet = \varnothing$. A net N is **deadlock-free** if $\forall t \in T \exists m \in [m_0\rangle \colon m[t\rangle$, and $m \neq m_f$. A net N **terminates properly** if a final marking is reachable from all reachable states $\forall m \in [m_0\rangle \colon m_f \in [m\rangle$.

B. Event logs

Process discovery techniques allow generating process models from event logs containing information on executed actions. In a simple case, event logs may contain actions names and a corresponding implementation order. We can augment this record with a timestamp (when an action occurs) and executor (what agent implements it).

Definition 3: Let $\mathcal{N}$ be a set of action names and $\mathcal{E}$ be a set of agent names. An **activity** is a triple (n, e, t), where $n \in \mathcal{N}$, $e \in \mathcal{E}$, and t corresponds to a timestamp. The set of all activities is denoted by Act. A **trace** $\sigma \in Act^+$ is a sequence of activities. An **event log** L is a multiset over Act^+, $L \in m(Act^+)$.

Different traces can be combined to form a case corresponding to a process implementation scenario. XES is a standard representation format adopted by IEEE [24] for logging events and processing them via process mining tools.

IV. Compositional process discovery

A. General outline

To support the compositional discovery of models from event logs generated by multi-agent systems, we assume a record of each action has a corresponding label of an agent

TABLE I
Fragment of an event log

Trace ID	**Action ID**	**Timestamp**	**Executor**
Trace 1			
	t_1	2017-03-01T17:23:40	Agent 1
	e_2	2017-03-01T19:12:05	Agent 2
	...	...	...
Trace 2			
	e_3	2017-03-02T21:13:47	Agent 2
	t_1	2017-03-04T21:14:40	Agent 1
	...	...	...

implementing it. The procedure of the compositional synthesis includes several steps to be implemented:

1) capturing a complete event log L from multi-agent system operation;
2) filtering the event log L by agent labels and producing a set of event logs L_e ($|L_e| = |\mathcal{E}|$), each trace in L_e consists of actions implemented by e only;
3) discovering a model for each agent separately from a set of event logs L_e;
4) defining interface pattern which describes how agents intercommunicate;
5) composing agent models and producing a multi-agent system model.

The step of defining interface pattern for agent interaction is implemented manually so far. We rely on an expert view on how agents should intercommunicate.

B. Software overview

A wide range of process discovery tools is implemented within the context of the open-source project ProM [25] continuously improving nowadays. However, there also exist many commercial tools using process mining approach to analyze and improve business process. They include Disco [26], QPR ProcessAnalyzer [27, pp. 11-12], myInvenio [28] to name but a few. Contrary to ProM, they provide more business-related solutions for process performance analysis and further improvement.

To process event logs we use the advanced ProM plug-in GENA [29] which allows to generate event logs with timestamps and originator labels as well as to augment logs with artificial events representing noise.

V. Composing Petri nets via interface patterns

This section provides a brief introduction to our approach to Petri net composition using interfaces and net morphisms.

A. Composing Petri nets via morphisms

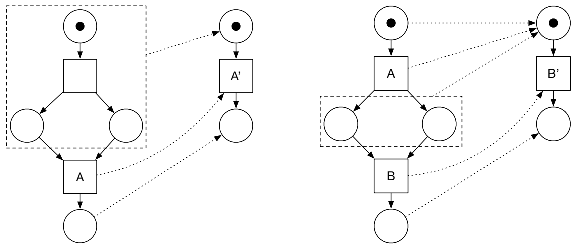
The notion of ω-morphism on Petri nets was first introduced in [3] for net systems and can be applied for safe nets.

Definition 4: Let $N_i = (P_i, T_i, F_i, m_0^i, L_i)$ for $i = 1, 2$ be two Petri nets, $X_i = P_i \cup T_i$. The ω-morphism is a total surjective map $\varphi : X_1 \to X_2$ such that:

1) $\varphi(P_1) = P_2$.

2) $\forall p_1 \in P_1 : m_0^1(p_1) > 0 \Rightarrow m_0^2(\varphi(p_1)) = m_0^1(p_1)$.
3) $\forall t_1 \in T_1 : \varphi(t_1) \in T_2 \Rightarrow \varphi(^\bullet t_1) = {}^\bullet\varphi(t_1)$, $\varphi(t_1{}^\bullet) = \varphi(t_1)^\bullet$.
4) $\forall t_1 \in T_1 : \varphi(t_1) \in P_2 \Rightarrow \varphi(^\bullet t_1{}^\bullet) = \{\varphi(t_1)\}$.

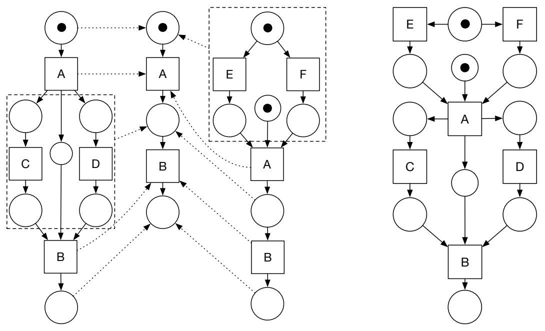
Figure 3 helps to explain requirements 2 and 3 of the definition, i.e. how transitions of N_1 can be mapped onto places and transitions of N_2.



(a) $\varphi(A) = A' \in T_2$ (b) $\varphi(A) = p \in P_2$

Fig. 3. Transition map options for ω-morphisms

To use morphisms for Petri net composition, we need to define morphisms from agent nets towards an interface which describes how they intercommunicate. Then we merge transitions having common labels and images. Figure 4 shows how two Petri nets are composed via ω-morphisms represented as dotted arrows.



(a) Defining ω-morphisms (b) Composition

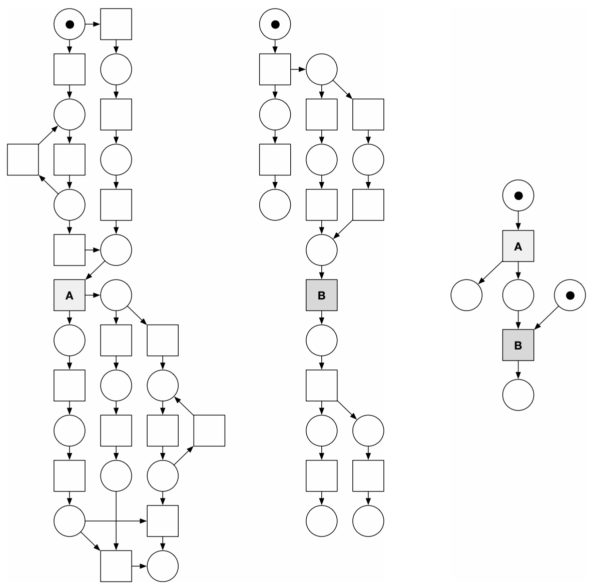
Fig. 4. Composing two Petri nets via ω-morphisms

As it was proved in [19], the use of morphisms allows preserving properties of interacting components in a composed process net. A composition obtained via ω-morphisms is deadlock-free and properly terminates iff source component nets and an interface net are deadlock-free and terminate properly as well.

B. Compositional interface patterns

To facilitate Petri net composition, we use compositional patterns for typical interfaces [2]. One of such patterns called the simple causality is schematically shown in Fig. 1, and Fig. 5 provides its instantiation. A pattern includes component and interfaces nets which might be merged according to the morphism composition rules if there is a need to produce a complete process model for comprehensive simulation.

It also has to be mentioned that to preserve concurrency in the implementation of interacting agents we expand interface nets with additional places and transitions keeping them weakly bisimilar with original interfaces. Consequently, extended interfaces allow us to obtain composition results with the clearly identified behavior of each component.



(a) Behavior of agents (b) Interface

Fig. 5. Instantiating simple causality pattern

Figure 5(b) shows how we have expanded interface net for this pattern. We use expanded interfaces only for our inner purposes. The end user does not need to know the underlying theoretical aspects of our approach.

VI. Some experimental evaluation

In this section, we describe a preliminary experiment on using the simple causality pattern for compositional process discovery.

A. Processing event logs

Using GENA and the composition result obtained from the instantiated simple causality pattern (see Fig. 5) we have generated the event log with 3000 traces. Then we have filtered the initial log by executors using ProM. The obtained event logs have the characteristics presented in Table II. Generation results for Agent A show bigger values due to cycles.

TABLE II
Characteristics of the event logs

	Log L	**Log** L_A	**Log** L_B
Number of traces	3000	3000	3000
Number of events	58466	34466	24000
Events per trace (min)	17	9	8
Events per trace (max)	43	35	8
Events per trace (mean)	19	11	8

B. Discovering a system model from log L

Figure 6 shows the fragment of the Petri net discovered from the event log L using Inductive Miner and ProM. The behavior of agents is distinguished by colors.

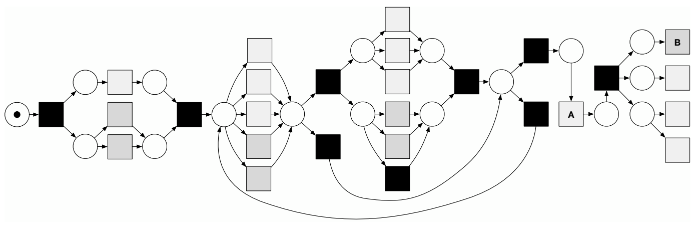


Fig. 6. The fragment of the system model discovered from L

This discovered model is quite well-structured (constructed out of clear blocks) but it does not allow to identify the behavior of different agents. That is why, it is hard to yield the complete picture of agent intercommunication scheme.

C. Discovering and composing models from logs L_A and L_B

Figure 7 shows the fragment of the merged Petri nets we discovered from the agent event logs L_A and L_B also using Inductive Miner and ProM. Note, that Petri nets discovered by Inductive Miner are always safe. Hence we can apply morphisms to compose discovered models of agents.

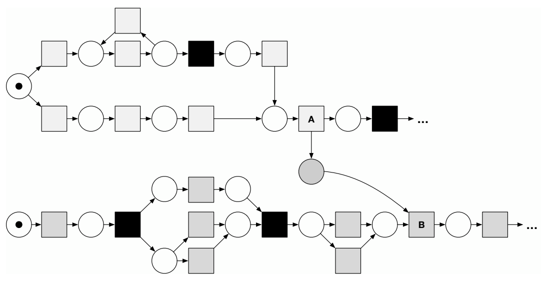


Fig. 7. The fragment of the composed model discovered from L_A and L_B

The merged model allows us to identify the behavior of agents clearly and how they intercommunicate. Using morphisms guarantees inheritance of properties such as deadlock-freeness and proper termination of agents by the entire net.

D. Analysis of the experiment results

ProM implementation of Inductive miner offers three configuration options:

1) event logs with **infrequent** behavior;
2) event logs with **incomplete** behavior;
3) event logs with **lifecycle** events (start/finish of events);
4) exhaustive k-successor algorithm.

We do not work with incomplete logs or with lifecycle logs for now. So, in our experiment we have discovered models of system and agents shown in previous subsections in accordance with options 1 and 4 and compared them using structural process discovery metrics.

Table III provides the comparison of structural characteristics for the directly discovered and composed system models. We compared obtained models with respect to the number of Petri net elements and structure metric which assess the overall complexity of a model by breaking it into trivial constructs and assigning weights to each reducing step. Models discovered with infrequent configuration are denoted as INFR, models discovered with exhaustive configuration are denoted as EXHS.

TABLE III
STRUCTURAL ANALYSIS OF SYSTEM MODELS

		Direct		Composition	
	Source	INFR	EXHS	INFR	EXHS
Places	28	30	47	35	39
Transitions	27	44	46	40	41
Arcs	68	100	114	89	93
Structuredness	9360	240	496	872	1208

The experiment results show the increase in transition numbers because of adding silent transitions. Compositional patterns obviously decrease a number of arcs, compared to direct discovery, as long as we simplify agent intercommunication. Composed models also preserve complex control flows as shown by structuredness measure. Separately discovered agent models and their composition exhibit more precise cycle discovery.

We have also conducted conformance checking for directly discovered and composed models. As it was mentioned above, there are four standard quality dimensions, namely fitness, precision, simplicity, and generalization. Simplicity is analyzed above via structural analysis. We do not estimate generalization since there are no complex cyclic or concurrent constructs to instantiate the simple causality pattern. Table IV shows values obtained for fitness and precision of discovered and composed system models.

TABLE IV
QUALITY ANALYSIS OF SYSTEM MODELS

		Direct		Composition	
	Source	INFR	EXHS	INFR	EXHS
Fitness	1,0000	1,0000	0,9684	1,0000	1,0000
Precision	0,6992	0,3631	0,5508	0,5629	0,6232

Both discovered and composed system models preserve the appropriate level of fitness, the composition does not block its preservation. What is more important, using compositional patterns produces models with precision nearer to that of the source model compared to direct discovery results. Composed models approximately 30% more precise than discovered ones.

To sum up, we used the simple causality pattern to produce the model of the multi-agent system. Assessment results showed that the composed models are highly competitive with the models directly discovered from complete event logs in the context of their relative structural complexity evaluations and conformance checking results.

VII. Conclusion and future work

In this paper, we have proposed the solution to the problem of discovering structured models for the processes with several participants (agents). The key idea is to automatically obtain the correct and complete process models from the separate source models of its components. The interaction between agents is defined by experts.

To prove the correctness of the composition we adopt the approach based on Petri net morphisms. We refer to the compositional patterns proposed for the correct synthesis of models for multi-agent processes. In the context of this work, we conducted the preliminary experiment on using the simple causality pattern for constructing the complete model from discovered agent models. The analysis of experimental results (conformance and complexity) showed that composed models are highly competitive compared to the models obtained directly. Moreover, our compositional approach to process discovery allows producing models with the clearly identified behavior of interacting agents.

We aim to continue developing of compositional patterns for typical interfaces and providing experimental process discovery implementations for them using also real-live event logs. Also, we will proceed with complex synchronization patterns with relations on action sets and their correct combinations.

VIII. Acknowledgments

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No.16-01-00546.

References

[1] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Springer Publishing Company, Incorporated, 2011.

[2] R. A. Nesterov and I. A. Lomazova, "Compositional process model synthesis based on interface patterns," in *Tools and Methods of Program Analysis, TMPA-2017. Proceedings*, ser. Communications in Computer and Information Science. Springer International Publishing, 2017, (to be published).

[3] L. Bernardinello, E. Mangioni, and L. Pomello, "Local state refinement and composition of elementary net systems: An approach based on morphisms," *Trans. Petri Nets and Other Models of Concurrency*, vol. 8, pp. 48–70, 2013.

[4] J. Buijs, B. Dongen, and W. Aalst, "On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery," in *OTM Federated Conferences, 20th International Conference on Cooperative Information Systems (CoopIS 2012)*, ser. Lecture Notes in Computer Science, vol. 7565. Springer-Verlag, Berlin, 2012, pp. 305–322.

[5] K. B. Lassen and W. M. P. van der Aalst, "Complexity metrics for workflow nets," *Inf. Softw. Technol.*, vol. 51, no. 3, pp. 610–626, 2009.

[6] L. Wen, W. M. Aalst, J. Wang, and J. Sun, "Mining process models with non-free-choice constructs," *Data Min. Knowl. Discov.*, vol. 15, no. 2, pp. 145–180, Oct. 2007.

[7] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs containing infrequent behaviour," in *Business Process Management Workshops - BPM 2013 International Workshops, Beijing, China, August 26, 2013, Revised Papers*, 2013, pp. 66–78.

[8] A. J. M. M. Weijters, W. M. P. Van Der Aalst, and a. K. A. D. Medeiros, "Process Mining with the HeuristicsMiner Algorithm," *BETA Working Paper Series WP*, vol. 166, 2006. [Online]. Available: http://cms.ieis.tue.nl/Beta/Files/WorkingPapers/Beta_wp166.pdf

[9] S. J. van Zelst, B. F. van Dongen, and W. M. P. van der Aalst, "Ilp-based process discovery using hybrid regions," in *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015*, 2015, pp. 47–61.

[10] C. W. Günther and W. M. P. van der Aalst, "Fuzzy mining - adaptive process simplification based on multi-perspective metrics," in *Business Process Management, 5th International Conference, BPM 2007, Proceedings*, 2007, pp. 328–343.

[11] W. M. P. van der Aalst and C. W. Gunther, "Finding structure in unstructured processes: The case for process mining," in *Proceedings of the Seventh International Conference on Application of Concurrency to System Design*, ser. ACSD '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–12.

[12] J. C. A. M. Buijs, "Flexible Evolutionary Algorithms for Mining Structured Process Models," Ph.D. dissertation, Eindhoven University of Technology, 2014.

[13] J. D. Smedt, J. D. Weerdt, and J. Vanthienen, "Multi-paradigm process mining: Retrieving better models by combining rules and sequences," in *On the Move to Meaningful Internet Systems: OTM 2014, Amantea, Italy, October 27-31, 2014, Proceedings*, 2014, pp. 446–453.

[14] J. de San Pedro and J. Cortadella, "Mining structured petri nets for the vizualization of process behavior," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16. New York, NY, USA: ACM, 2016, pp. 839–846.

[15] A. A. Kalenkova, I. A. Lomazova, and W. M. P. van der Aalst, *Process Model Discovery: A Method Based on Transition System Decomposition*. Cham: Springer International Publishing, 2014, pp. 71–90.

[16] A. A. Kalenkova and I. A. Lomazova, "Discovery of cancellation regions within process mining techniques," *Fundam. Inform.*, vol. 133, no. 2-3, pp. 197–209, 2014.

[17] W. Reisig, *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[18] G. Winskel, "Petri nets, morphisms and compositionality," in *Advances in Petri Nets 1985, Covers the 6th European Workshop on Applications and Theory in Petri Nets-selected Papers*. London, UK, UK: Springer-Verlag, 1986, pp. 453–477.

[19] L. Bernardinello, E. Monticelli, and L. Pomello, "On preserving structural and behavioural properties by composing net systems on interfaces," *Fundam. Inf.*, vol. 80, no. 1-3, pp. 31–47, Jan. 2007.

[20] L. Bernardinello, L. Pomello, and S. Scaccabarozzi, "Morphisms on Marked Graphs," in *International Workshop on Petri Nets and Software Engineering (PNSE'14)*, vol. 1160, 2014, pp. 113–127.

[21] I. A. Lomazova, "Interacting workflow nets for workflow process re-engineering," *Fundam. Inform.*, vol. 101, no. 1-2, pp. 59–70, 2010.

[22] Y. Cardinale, J. E. Haddad, M. Manouvrier, and M. Rukoz, "Web service composition based on petri nets: Review and contribution," in *Resource Discovery - 5th International Workshop, RED 2012, Heraklion, Greece, May 27, 2012, Revised Selected Papers*, 2012, pp. 83–122.

[23] R. Hamadi and B. Benatallah, "A petri net-based model for web service composition," in *Proceedings of the 14th Australasian Database Conference - Volume 17*, ser. ADC '03. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2003, pp. 191–200.

[24] "XES (eXtensible Event Stream." [Online]. Available: http://www.processmining.org/logs/xes

[25] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst, *The ProM Framework: A New Era in Process Mining Tool Support*. Springer Berlin Heidelberg, 2005, pp. 444–454.

[26] C. W. Günther and A. Rozinat, "Disco: Discover your processes," in *BPM (Demos). CEUR Workshop Proceedings*, vol. 940, 2012, pp. 40–44.

[27] I.-M. Ailenei, "Process Mining Tools: A Comparative Analysis," Master's thesis, Eindhoven University of Technology, 2011.

[28] "Business Process Management Tool — myInvenio." [Online]. Available: https://www.my-invenio.com

[29] I. S. Shugurov and A. A. Mitsyuk, "Generation of a Set of Event Logs with Noise," in *Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014)*, 2014, pp. 88–95.

An online tool for requirements engineering, modeling and verification of distributed software based on the MDD approach

Daria Lozhkina,
Sergey Staroletov
Polzunov Altai State Technical University,
Lenin avenue 46, Barnaul, 656038, Russia
Email: serg_soft@mail.ru

***Abstract*—The existing verification tools are so complex and intended for experts in this area, so it can be a significant obstacle to the practical use of them. While these tools have many advantages, they have not been widely used. For the purpose of automatic verification, properties of complex systems can be expressed in Linear-time Temporal Logic, and the model of the system is described in a special language. In this context, there is a problem to use it by the software engineers. This problem led us to the idea of developing an MDD online tool for modeling distributive software that allows many users to describe their systems and set the specification for it by using natural language and visual notations and then verify the created model.**

Keywords*—MDD, verification, requirements engineering, LTL, Spin

I. Introduction

As the software products become more complex, a problem of their correctness and safety is coming up, especially when we develop high-risk, vital software or distributive software with an amount of highly-interoperable components. In some cases, it is possible to develop a product, deploy it and run to find software bugs under some test cases. Software testing is widely used in practice, but it is clearly not possible to prove the correctness of the system by such an approach, that ambiguous makes usage of the software in vital areas unacceptable, the cost of undetected bugs can be too high.

Our paper is devoted to another solution, a formal verification. It is an important method for improving the quality of complex systems. The developer must specify system requirements, and a formal verifier automatically ensures that the product satisfies them. The approach provides a powerful way to detect errors that are often subtle and difficult to reproduce. A formal verification of software programs proves that the model of a program satisfies to a formal specification of its behavior. To perform a process of automatic verification, the software engineer must describe the model of the system and formulate the properties expressed in a calculus language. Here we have the two main problems - how to build an adequate model and how to collect all the requirements to check?

This paper considers that the properties are expressed in Linear-time Temporal Logic [1]. However, a syntax of LTL is complicated and can be a significant obstacle to the practical use of verification tools which works with LTL predicates.

For example, according to the specification the system must always be in the following state: if $stateP$ becomes true, sometimes in the future $stateQ$ will be true, and $stateP$ will remain true until $stateQ$ becomes true [2]. This requirement is expressed in LTL language as a formula:

$$G(p \to (pUq))$$

Here G is a temporal operator "Global"("Always"), and U is a temporal operator "Until".

It is rather difficult to translate the natural language of system requirements to an LTL approach. Some formulas can be very long and complex; even the experts cannot express the properties without troubles.

Our solution to this problem to develop a web portal is devoted to building a model of the checked system and collect all the requirements for it in one place. We use visual notation to express temporal operators graphically. We assume it is a right approach to help software engineers to set the requirements as they usually construct some flowcharts. We propose an MDD (Model-driven development [3]) solution to describe a model of a software system for further verification and implementation.

II. Software Testing vs Verification

We need a lot of procedures to ensure that a product meets its quality goals and satisfies the specified functional requirements. In this case, software developers use verification and validation processes and software testing. Only the combination of these procedures can improve the system quality.

Software testing today is a widely used process in software engineering to executing an application with the intent of finding the software bugs (errors or other defects) under some test cases. However, testing under all the combinations of inputs and initial states is not feasible, it can find only some bugs and cannot be used as conformity checking of the system to given requirements in all cases. Even for single-threaded application with respect to Turing's Halting problem we cannot prove the absent of errors. Especially, it is a fundamental problem with testing of parallel and distributed communication systems. The developer is not able to control all the possible combinations of interactions between parallel processes, thread scheduling and interleaving, common resources sharing.

So we need a way to prove that the model of software meets all the requirements, and we need a formal verification. This process is much more complicated than testing, and it requires specialized knowledge.

The software testing process must be used in association with verification and validation. Testing and verification are the pieces of checking process to ensure that a software system meets its specifications and that it fulfills its intended purpose [4]. However, they are not the same thing.

Briefly, *software verification* is ensuring that the product conforms to its specifications, while *software validation* ensures that the product meets the customer's actual needs. The most well-known questions to distinct these terms (they were expressed by Boehm [5]):

- Validation: Are we building the right product?
- Verification: Are we building the product right?

The verification generally comes before the validation and can catch errors that validation cannot catch. Therefore, the goal of the software verification is to assure that a product fully satisfies all the expected requirements.

The main problem of verification today - is a big amount of different academic methods and lack of popular tools in the software engineering community.

III. Requirements engineering

The process of requirements engineering includes many activities, from goal elicitation to requirements specification. The aim is to develop an operational requirements specification that is guaranteed to satisfy the goals [6].

To build a good and reliable software the customers, software engineers and business analytics have to hard work together. They must produce a specification where all the requirements will be reflected. But it is hard to do, because even the customer doesn't know exactly what does he need.

Our goal is to provide a place to model the system properties online, to collaborate customers and engineers to build requirements to complex systems with properties expressed in a visual notation of logic formulas we have created (see Figure 1).

IV. Model based development

MDD [3] is a common name for techniques to build software starting from the certain model. The Model could be a static one to describe class-to-class relations and a dynamic one to describe activities and parallel processes inter-operations.

Well known UML diagrams changed a way of software development process, to develop a software based not on code, but on a model. In the past, methodologies were called in *OP style (like Object-Oriented Programming, Aspect-Oriented Programming) and were related to a ”programming”, but now in the software engineering, we see that *DD methodologies are upcoming which relate now to a ”development” (”development” is much stronger word than ”programming”).

The goal of *DD methodology is something that encourages us to write code. In the MDD, a model is a way to write

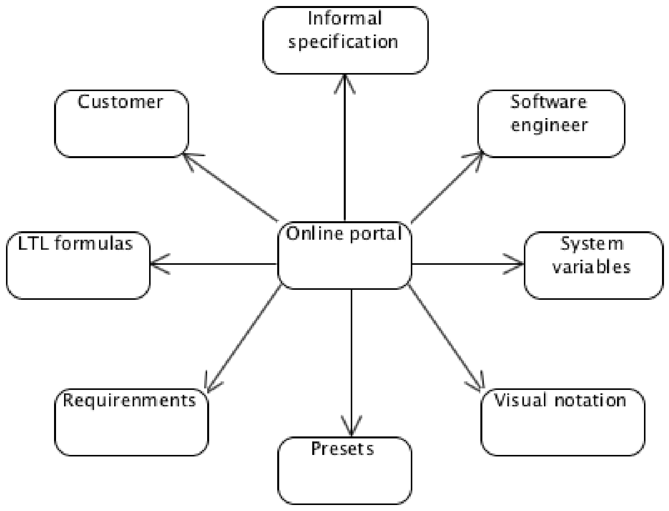


Fig. 1. Requirements engineering as links between customer, engineer, portal and tools

code and even not to write but to get the most of the code from the model and then to verify it.

In the paper [7] we created a link from the MDD process to the verification process by generating and checking the model of a system as modified agent-based finite automatons, provided ways to transform the model to a formal verifier to check the properties.

Now the aim of the research is to improve the MDD process and combine it with the requirements engineering and Model-based checking approach.

V. Model-based checking

Model-based checking in general is an automated technique for verifying finite-state concurrent systems to check whether the software meets system requirements formulated at the initial stage [8] or to provide a counterexample.

There are several steps in verifying the correctness of the system [8]:

1) To specify the properties that prescribe what the system should do, and what should not do.
2) To construction a formal model for the system that addresses how the system behaviour.
3) To perform the verification process.

VI. Verification tools problem

Today a lot of model checking tools (usually called verifiers) have been developed by the research institutions. The tools are powerful but very complex, and programming shells for them are inconvenient. Most of them use specific functional languages to describe the input model.

Spin (Simple Promela Interpreter) is a popular open-source software verification tool designed specially to model distributed and parallel software.

To use SPIN, the software engineer needs to know following languages:

1) A high level functional language *Promela* (short for: PROcess MEta LAnguage) [9] to specify software system's description as a model.
2) A *Linear-time Temporal Logic* [1] to express the system requirements. Many of the system properties can be expressed and verified without the use of LTL, but most of them must be.

Now we are targeting to Promela language, but in future, it is possible to extend the software with some target languages and verifiers.

As we develop an online portal, the user is not needed to install any software to do the verification. The verifier checker can run on the server to verify the model generated from the user's diagrams.

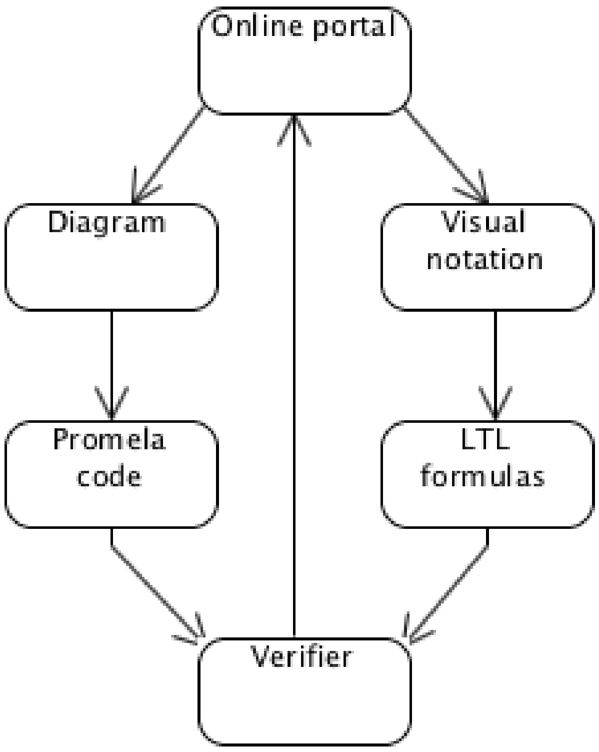


Fig. 2. The online portal life-cycle

VII. A verification process

We are working on verification automation process to allow the portal to be a bridge between a user and a verifier. An executable model code is automatically generated from the constructed diagram, and then a verifier on the server is called on it. We use the Spin to perform the verification process. For this, the model diagram is translated into Promela language, and for each of the formulas given by the user the LTL Never Claims checking are executed. The output stream of the verifier is collected, parsed and displayed to the user on the site. If there were no errors, the user will receive the verdict "No errors found", otherwise the verifier will run in the simulation mode with the counter-example as an error trail. Then the output stream of verifier will contain an information about the incorrect sequence of work, and the user will receive the data about the sequence of messages in the form of a diagram indicating the wrong steps.

VIII. A Visual Notation for Boolean and Temporal operators

To provide a better understanding of LTL formulas for humans, we use a visual notation, different to an existing approach [10].

The Model-based checking tools perform the Temporal Logic translation of the model as the sequence of system states. A formula of Temporal Logic describes the set of sequences in time for which it is true. A state in our solution is a set of given system variables values.

Each user-defined *state* is depicted as a single block that can connect with other states with Temporal operators.

All the operators are represented by a circle that contains a special symbol (see Table I).

TABLE I. Our visual notation for LTL operators

Operator	**LTL Formula**	**Promela syntax**	**Visual Notation**
State	State	State	State
And	State1 $\wedge$ State2	State1 && State2	State 1 Ⓐ State 2
Or	State1 $\vee$ State2	State1 || State2	State 1 Ⓞ State 2
Xor	State1 xor State2	State1^^State2	State 1 Ⓧ State 2
Implication	State1 $\rightarrow$ State2	State1 -> State2	State 1 → State 2
Equivalence	State1 $\equiv$ State2	State1 <-> State2	State 1 ≡ State 2
Negation	$\neg$ State	!State	¬ State
Globally	G State	[]State	∞ State
Finally	F State	<>State	✳ State
Until	State1 U State2	State1 U State2	State 1 ▷ State 2
Parenthesis	(State1 $\wedge$ State2)	(State1 && State2)	State 1 Ⓐ State 2

For classical Boolean operators it is the proper initial letter (**A** for *And*, **O** for *Or*, **X** for *Xor*) or the common symbol (for *Implication*, *Negation* and *Equivalence*). As our target was to develop a visual notation for an LTL expression as clear as possible, we developed the following visual elements for representation of temporal operators:

- *Globally* is depicted with an infinity symbols, ideally representing the event that has to hold during all the times.
- *Finally* means that the state eventually has to hold, we propose an asterisk as a symbol of new event.
- *Until* is represented with the triangle touching the line, that means the fact that the first event has to hold at least until the second one, which holds at the current or future time.
- *Parenthesis* are necessary for specifying evaluation property in formula, also we can surround existing sequences of states and operators into one block and

use it as a single unit, so we represent it as a rounded rectangle. Thereby it is possible to translate complex formulas.

In case of unary operators (*Globall*, *Eventually*, *Negation* and *Equivalence*) the circle attaches the one operand on one side and in case of binary operator (*And*, *Or*, *Xor*, *Implication*, *Until*), the operator attaches to the both operands.

IX. Implementation of our Online Tool

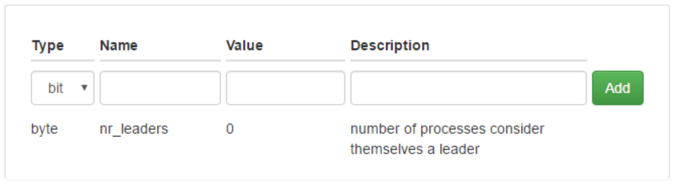
The application allows users to create an account, add, store projects and edit them. Preparation for the verification process consists of two steps.

A. *Creating the model*

The first one is to create the model of a system. This tool allows a user to declare global variables, processes and channels for communication between them. The user interface of the model-creating tool contains several panels.

The top tool consists of four panels, which can be hidden. One of them is intended to edit project properties (e.g., title, description).

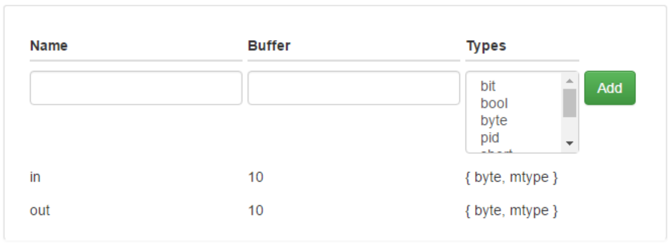
The *Global Variables Panel* (see Figure 3) allows a user to add system variables as follows: choosing a type with using the selected list contains Promela types and entering a name, an initial value and a description of a variable. On the bottom of this panel, a user can see the complete list of the variables. The frame of this panel is used as a template for the entire application.



Type	Name	Value	Description	
bit ▾				Add
byte	nr_leaders	0	number of processes consider themselves a leader	

Fig. 3. The Global Variables Panel

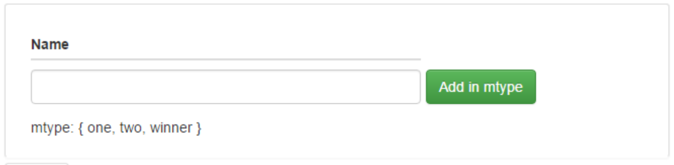
The channel (models a Promela's channel) initializer specifies a channel capacity, as a constant, and the structure of the messages that can be stored here, as a comma-separated list of type names. Therefore, the *Channel Panel* (see Figure 4) looks like a Variables Panel and allows creating the channel with necessary parameters: there are fields for name, buffer size and a selection list contains all possible system types.



Name	Buffer	Types	
		bit bool byte pid	Add
in	10	{ byte, mtype }	
out	10	{ byte, mtype }	

Fig. 4. The Channel Panel

Also, the syntax of the Promela language includes mtype declaration for defining symbolic names, so the top panel contains a special *Message Type Panel* (see Figure 5) to set the possible message types. Chan and mtype are not available in the list of the possible system types in other panels.



Name

Add in mtype

mtype: { one, two, winner }

Fig. 5. The Message Type Panel

The main panel is a tabbed field, where each tab contains a model of a particular process (see Figure 6). It consists of the board for drawing a diagram of a process, the palette with a set of visual primitives and text fields for the name and local variables. Set of the visual primitives contains the standard shapes: circles for Start and End states, a simple rectangle for Step, a diamond for If and Switch operators and a special symbol to start the process. Processes are able to interactive with messages, so we propose to represent primitives for message sending and receiving processes with rectangles contain light or dark symbol of mail on the top-left corner. A user can drag objects from the panel to the board, change the text and connect the shapes with arrows to model their order.

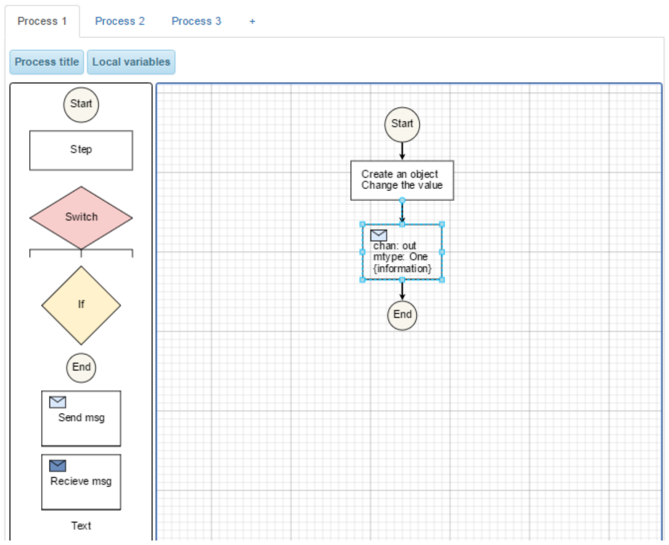


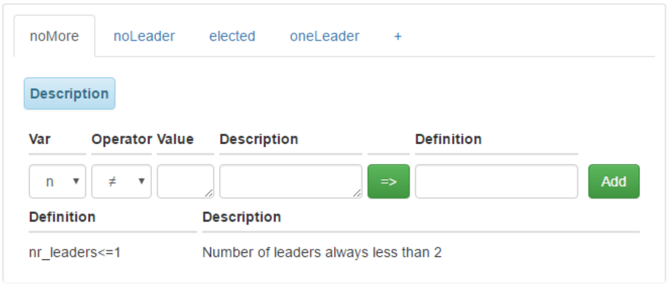
Fig. 6. The Main Panel

When user finished declaring the variables and created the diagrams that represented processes, the software can generate the runnable code of the system in Promela language to do further verification.

B. *Setting requirements*

The second step is to declare system requirements.

The top panel contains a tabbed field with template panel that was mentioned before (see Figure 7). Each tab is intended to declare and describe one state; variables properties are defined as follows: the system variables and the possible operators are available to the user as selection lists, so the user can choose them, set the values, enter the description and push the button to add it to the current state. There is the complete list of the states of the bottom of the tab. The next panel shows the created code so that the user can look on it to create desired requirements.



Var	Operator	Value	Description	Definition
n	≠			

Definition	Description
nr_leaders<=1	Number of leaders always less than 2

Fig. 7. The State Panel

The interface of the main panel is similar to the Process one (see Figure 8). It is also a tabbed field for the requirements, consists of the drawing board and panels with the user-defined states and logic operations, allows dragging objects from the panels to the board and connecting them. It is also possible to set a name, definition and description for each requirement. We are able to parse this diagram and translate it to an LTL formula according to our visual notation and Promela syntax.

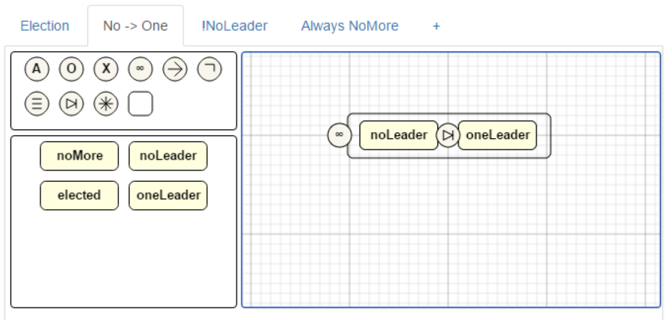


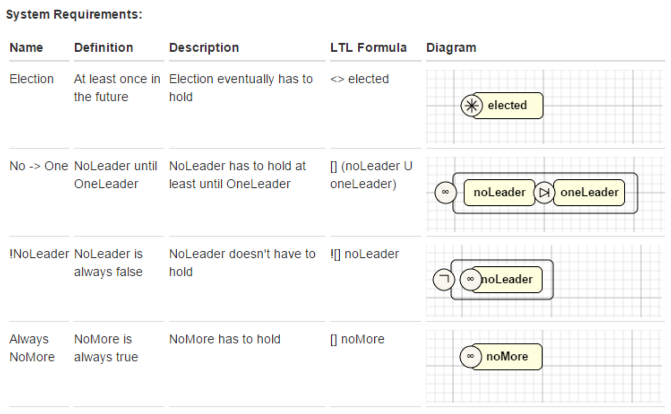
Fig. 8. The Requirement Panel

The bottom panel shows the complete list of the system requirements as a table with the following fields: name, definition, and description of the requirement, an LTL formula that was generated from the diagram, and an image representation of this one (see Figure 9).

Finally, as a result of the MDD process, we have all necessary data for further verification: Promela code and LTL formulas.

X. Practical Use

Last year we developed a replicated chat using the Leader Election Algorithm as the problem of reliable distributive communication [11] . Leader's election is the process of designating a single process as the organizer of some task distributed among several computers (nodes) [12]. In other words, there are some distributed nodes with the special weights (under the word weight we can specify any information about this node, e.g. the size of the log, the id, the special number etc.). The nodes interact with each other asynchronously. The



System Requirements:

Name	Definition	Description	LTL Formula	Diagram
Election	At least once in the future	Election eventually has to hold	<> elected	elected
No -> One	NoLeader until OneLeader	NoLeader has to hold at least until OneLeader	[] (noLeader U oneLeader)	∞ noLeader oneLeader
!NoLeader	NoLeader is always false	NoLeader doesn't have to hold	![] noLeader	¬ ∞ noLeader
Always NoMore	NoMore is always true	NoMore has to hold	[] noMore	∞ noMore

Fig. 9. The List of the requirements

problem is to find the node that will be the global leader for the entire system. During our research work, we analyzed some special algorithms: Paxos [13], Leader Election in rings[14] and others. As these algorithms are intended for distributed systems and the states of processes after some steps only depend on the states of neighbor nodes, their work has to be verified.

Now it is easy to check the correctness of our developed distributive chat with using our online portal.

As an example, we can do it for the Leader Election Algorithm in unidirectional ring. Firstly, We need to design a model of this algorithm to generate a Promela code. Each node executes the process, which updates the parameter values on the active nodes. We added the global variables (see Figure 3), channels (see Figure 4) and message types (see Figure 5). Also, we designed the model using the diagram-builder tool (see Figure 10).

The second step is to set states and define requirements (see Figure 9):

1) Leader election eventually has to hold.
2) The state, in which there is no leader, has to hold at least until a leader will be elected.
3) There is always a leader in the system.
4) It is not possible to elect more than one leader.

Therefore, now we can perform modeling and then the verification process and ensure that the algorithm is correct without having special software on the user's computer and knowledge of the specialized languages.

XI. Conclusion and Future Work

The described online tool has been implemented in PHP and JS programming languages with using Symfony MVC framework.

We are going to deploy the portal to our domain [15].

Next steps are:

- some pre-defined templates of LTL formulas;
- sample projects for well known distributive algorithms;

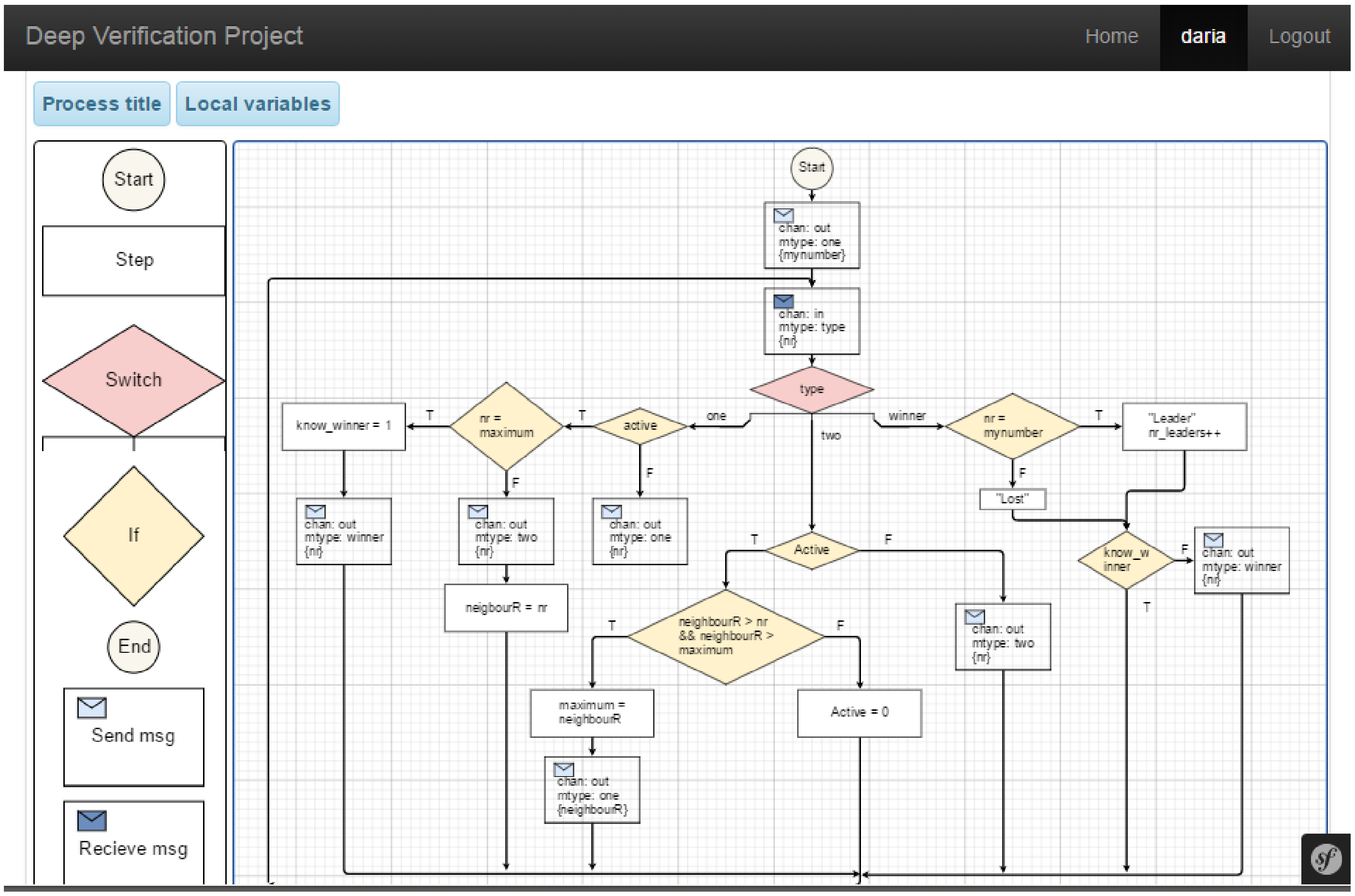


Fig. 10. The model of the Leader Election algorithm

- generate not only Promela code from the model to verify it but also an Erlang code to run (because Erlang and Plomela are actor-based languages);
- think about ability of distributed running the verifier process on the server side and balance the load;
- think to a better way to represent multiple processes into one window;
- examine ways of getting the requirements from the informal text of specification and convert it to visual notation.

The results partially were obtained within the RFBR grant (project No. 17-07-01600).

REFERENCES

[1] Linear-Time Temporal Logic. http://www-step.stanford.edu/tutorial/temporal-logic/temporal-logic.html

[2] Model Checking: A Complement to Test and Simulation. http://www.cse.chalmers.se/

[3] T. Stahl, M. Voelter. Model-Driven Software Development: Technology, Engineering, Management (book, ISBN 0470025700)

[4] Verification, Validation & Certification. https://users.ece.cmu.edu/ koopman/des_s99/verification/index.html

[5] Boehm B.W., Software engineering economics, *IEEE Trans. on Software Eng.*, Vol. 10, no. 1, Jan. 1984. pp. 4 - 21.

[6] D. Alrajeh, J. Kramer, A. Russo, S. Uchitel. Elaborating Requirements using Model Checking and Inductive Learning. IEEE Transactions on Software Engineering (Volume: 39, Issue: 3, March 2013). pp. 361-383

[7] S. Staroletov. Model Driven Developing & Model Based Checking: Applying Together / Tools & Methods of Program Analysis TMPA-2014:Kostroma, 2014. - pp. 215-220. ISBN 975-5-8285-0719-1. Presentation: http://www.slideshare.net/IosifItkin/model-driven-developingmodelbasedchecking

[8] Clarke E. M., Grumberg Orna, Peled Doron A. Model Checking. Cambridge, Mass : The MIT Press, Jan. 1999. p. 314.

[9] Spin - Formal Verification. http://spinroot.com

[10] M. Brambilla, A. Deutsch, L. Sui, V. Vianu. The Role of Visual Tools in a Web Application Design and Verification Framework: A Visual Notation for LTL Formulae. D. Springer-Verlag. ICWE 2005, LNCS 3579, pp. 557-568,

[11] Lozhkina D.D., Staroletov S.M. The application of the distributed leader selection algorithm for the problem of reliable communications. Abstracts at the All-Russian Scientific and Technical Conference of Students, Post-Graduates and Young Scientists ”Science and Youth - 2016”. Software engineering section. AltSTU, 2016, pp. 19-23 (in Russian), http://edu.secna.ru/media/f/pi2016.pdf

[12] D. Ongaro, J. Ousterhout. In Search of an Understandable Consensus Algorithm. Stanford University, 2014. p. 18.

[13] Lamport, Leslie. Paxos Made Simple. *ACM SIGACT News (Distributed Computing Column)*, Vol. 32, no. 4. 2001, pp. 51 - 58.

[14] H. Attiya and J. Welch, Distributed Computing: Fundamentals, Simulations and Advance Topics, *John Wiley & Sons inc.*, 2004.

[15] Deep verification project. http://deepverification.com/

On minimization of Timed Finite State Machines

Aleksandr Tvardovskii, Nina Yevtushenko
National Research Tomsk State University
Tomsk, Russia
tvardal@mail.ru, nyevtush@gmail.com

Abstract – **Finite State Machines (FSMs) are widely used as formal models for analysis and synthesis of components of control systems. In order to take into account time aspects, timed FSMs are considered. In this paper, we first discuss how timed models can be used when implementing controllers for various systems and then address the problem of minimizing a FSM with timed guards and input and output timeouts (TFSM) as the complexity of many problems of analysis and synthesis of digital and hybrid systems including high-quality test derivation significantly depends on the size of the system (component) specification. The behavior of a TFSM can be described using a corresponding FSM abstraction and a proposed approach for minimizing a TFSM is based on such abstraction. Moreover, we minimize not only the number of states as it is done for classical FSMs but also the number of timed guards and timeout duration. We show that for a complete deterministic TFSM there exists the unique minimal (canonical) form, i.e., a unique time and state reduced TFSM that has the same behavior as the given TFSM; for example, this minimal form can be used when deriving tests for checking whether an implementation under test satisfies functional and nonfunctional requirements.**

Keywords – Timed Finite State Machines; reduced form; minimal form

I. INTRODUCTION

Finite State Machines (FSM) are widely used for synthesis and analysis of components of digital and hybrid control systems [see, for example, 1, 2], especially when implementing controllers for various systems and processes [3, 4, 5]. The complexity of solving a number of synthesis and analysis problems significantly depends on the number of states of a FSM that represents the system (component) specification. In particular, almost all FSM-based test derivation methods [2, 6, 7] for telecommunication protocols and other control systems with deterministic behavior are developed for reduced FSMs, i.e., FSMs which have different behavior at any two different states. In the classical FSM theory, the FSM minimization methods are well developed, i.e., given a deterministic FSM, it is well known how to derive a reduced form of the FSM.

Nowadays time aspects become very important when describing the behavior of digital and hybrid control systems, and, respectively, classical FSMs were extended with time variables [see, for example, 7, 8, 9]. In this paper, we consider FSMs with timed guards and (input and output) timeouts (TFSM) which generalize TFSM models which have only timed guards or only timeouts [10, 11]. Timed guards describe the system behavior depending on a time instance when an input is applied. If no input is applied until an (input) timeout is expired then the system can spontaneously move to another state. An output timeout (delay) describes time that is needed for processing a given transition. In the model with timed guards and timeouts, we can also take into account that for example, pressing a button to execute an appropriate action can take some time.

The complexity of many problems of analysis and synthesis of digital and hybrid systems including derivation of high-quality tests significantly depends on the size of the system (component) specification. For example, to the best of our knowledge, most test derivation techniques are developed for minimal forms of FSMs [6, 12]. When minimizing classical FSMs (and other trace models) most attention is paid to minimizing the number of states (and possibly) actions of the machine under investigation. Differently from classical FSMs, a timed FSM can have several non-isomorphic state reduced forms [10, 11]. The reason is that for timed transition systems, time aspects should be also taken into account when minimizing a TFSM.

In this paper, we propose the minimal (canonical) form of a complete deterministic FSM with timed guards and timeouts; this minimal form is a reduced well-defined TFSM having the same behavior as the given TFSM. A method for deriving the minimal form of a deterministic complete TFSM is also proposed.

The structure of the paper is as follows. Section 2 contains the preliminaries for classical and timed FSMs. In Section 3, the related work on minimizing TFSMs is briefly described; this section also has a TFSM example for describing a part of the system for elevator management. In Section 4, we show how the behavior of a TFSM can be described using an appropriate FSM abstraction. Section 5 contains the minimization procedure for a deterministic complete TFSM. In Section 6, the notion of a well-defined TFSM is proposed and the minimal form of a TFSM is defined as a state reduced well-defined TFSM that has the same behavior as the given TFSM; we also show that for each deterministic complete TFSM the minimal form is unique (up to isomorphism). Section 7 contains experimental results. Section 8 concludes the paper.

II. PRELIMINARIES

The notion of a Finite State Machine (FSM) is used when describing the behavior of a system that moves from state to state under input actions and produces a predefined output response. Formally, a FSM is a 4-tuple $\mathsf{S} = (S, I, O, h_S)$ where I and O are input and output alphabets, S is a finite non-empty set of states and $h_S \subseteq (S \times I \times O \times S)$ is the transition (behavior) relation. A transition (s, i, o, s') describes the situation when an input i is applied to S at the current state s. In this case, the FSM moves to state s' and produces the output (response) o. In this paper, we consider complete and deterministic FSMs, i.e., for each pair (s, i) $\in S \times I$ there exists exactly one transition (s, i, o, s') $\in h_S$. *A trace* or an *Input/Output sequence* α/γ, written often as an *I/O* sequence, of the FSM S at state s is a sequence of consecutive input/output pairs starting at the state s. Given a trace α/γ, α is the *input projection* of the trace (input sequence) while γ is the corresponding *output projection* (output sequence), i.e., the output response of the FSM to sequence α of applied inputs. At each state of a complete deterministic FSM, there is exactly one output response for each input sequence.

Given complete deterministic FSMs S and P and their states s and p, states s and p are *equivalent* if output responses at these states coincide for each input sequence, i.e., FSMs S and P at states s and p have the same behavior. If states s and p are not equivalent then they are *distinguishable*. For two states of a single FSM, there is the same definition of equivalent states. Complete deterministic FSMs S and P are *equivalent* if for each state of FSM S there exists an equivalent state of FSM P and for each state of FSM P there exists an equivalent state of FSM S. If any two different states of FSM S are distinguishable then FSM S is (*state*) *reduced*. A reduced FSM P is a *reduced form* of FSM S if FSMs S and P are equivalent. It is known [1] that given a complete deterministic FSM, two reduced forms of this machine are isomorphic, i.e., the reduced form is a *canonical representation* of a complete deterministic FSM.

The equivalence relation induces a partition E of the set of states of a complete deterministic FSM. Two states of the same class of the partition E are equivalent; two states of different classes of partition E are distinguishable.

A FSM with timed guards and timeouts (TFSM) is a FSM annotated with a *clock*; such TFSM has input and output timeouts and input timed guards [9]. When an input timeout expires at state s, the TFSM can spontaneously move to another state. A good example is a mobile when the screen becomes dark if no button is pressed during appropriate number of time units, i.e., no input is applied. An output timeout describes how long an applied input is processed at a given state. Input timed guards describe the behavior at a given state for inputs which arrive at different time instances. Correspondingly, a TFSM is a 5-tuple S = (I, S, O, h_S, Δ_S) where I and O are input and output alphabets, S is the finite non-empty set of states, $h_S \subseteq (S \times I \times O \times S \times \Pi \times Z)$ is the *transition relation* and Δ_S is the timeout function. The set Π is a set of *input timed guards*, and Z is a set of *output delays* which are nonnegative integers. The *timeout function* is the function Δ_S: $S \to S \times (N \cup \{\infty\})$ where N is the set of positive integers: for each state this function specifies the maximum time for waiting an input. An input timed guard $g \in \Pi$ describes the time domain when a transition can be executed and is given in the form of interval $\lceil min, max \rceil$ from [0; T), where $\lceil \in \{(, [\}, \rceil \in \{,),]\}$ and T is value of the (input) timeout at the current state. An output delay describes the number of ticks when an output has to be produced after applying an input. The transition (s, i, o, s', g, d) $\in S \times I \times O \times S \times \Pi \times Z$ means that TFSM S being at state s accepts an input i applied at time $t \in g$ measured from the moment when TFSM S entered state s; the clock then is set to zero and S produces output o after d time units counted from the moment when the input has been applied. Given state s of FSM S such that $\Delta_S(s) = (s', T)$, if no input is applied before the timeout T expires, the TFSM S moves to state s' and the clock is set to zero. If $s = s'$ then the clock is set to zero when timeout is expired. Given TFSM S, B_S is the largest finite boundary of timed guards.

For example, consider TFSM S in Figure 1. The output response of TFSM S at state s_0 to input i applied at time instance 0 is output o_1 with the delay of one tick, while if input i is applied at time instance 1, then S produces output o_2 with the delay of three ticks. If an input is not applied to TFSM S at state s_0 during 2 time units then S moves to state s_3 according to the timeout function.

Similar to [7], for each state s of TFSM S we consider the function $time(s, t) = s'$ that determines state s' that will be reached by the TFSM if no input was applied during t time units.

A *timed input* is a pair (i, t) where $i \in I$ and t is a real; a timed input (i, t) means that input i is applied to the TFSM at time instance t. A timed output is a pair (o, d) where $o \in O$ and d is the output delay. A sequence of timed inputs $\alpha = (i_1, t_1) \dots (i_n, t_n)$ is a *timed input sequence*, a sequence of timed outputs $\gamma = (o_1, d_1) \dots (o_n, d_n)$ is a *timed output sequence*. A sequence $\alpha/\gamma = (i_1, t_1)/(o_1, d_1) \dots (i_n, t_n)/(o_n, d_n)$ of consecutive pairs of timed inputs and timed outputs starting at the state s is a *timed I/O sequence* or a *timed trace* of TFSM S at state s. Similar to FSMs, α is an applied timed input sequence while γ is the corresponding output response of the TFSM to sequence α of applied inputs.

In order to determine the output response of the TFSM at state s to a timed input (i, t), state $s' = time(s, t)$ is calculated first. State s' is a state where TFSM moves from state s via timeout transitions such that the maximum sum Σ of all timeouts starting from state s is less than t. At the second step, a transition (s', i, o, s'', g, d) such that $t - \Sigma \in g$ is considered. According to this transition, the machine produces the output (o, d) to a timed input (i, t) applied at state s and moves to the next state s''. Thus, the output response of the TFSM to a timed input sequence at state s is iteratively determined starting from state s. Similar to FSMs, the set of all timed traces at all states of the TFSM determines the TFSM behavior.

TFSM S is a *deterministic complete* TFSM if for each two transitions (s, i, o_1, s_1, g_1, d_1), (s, i, o_2, s_2, g_2, d_2) $\in h_s$, it holds that $g_1 \cap g_2 = \varnothing$ and the union of all input timed guards at state s under input i equals [0; T) when $\Delta_S(s) = (s', T)$. In this paper, we consider only deterministic complete TFSMs.

Given complete deterministic TFSMs S and P and their states s and p, states s and p are *equivalent* if output responses at these states coincide for each timed input sequence. The notions of a (state) reduced TFSM, equivalent TFSMs and the partition into equivalent states are defined similarly to classical FSMs.

Two TFSMs S and P are *isomorphic* if there exists the one-to-one correspondence H: $S \to P$ such that there exists a transition $(s, i, o, s', g, d) \in h_s$ if and only if there exists a transition $(H(s), i, o, H(s'), g, d) \in h_p$ and $\Delta_P(H(s)) = (H(s'), T)$ if $\Delta_S(s) = (s', T)$.

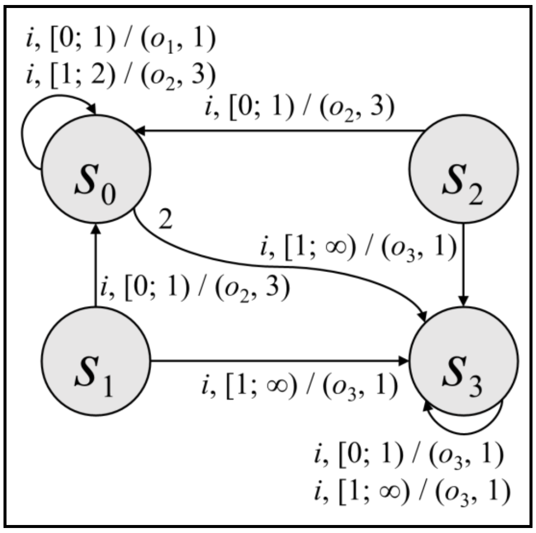


Fig. 1. TFSM S

III. Related work

There is a big body of work for minimizing classical deterministic (complete and partial) FSMs. An interested reader can be referred to [1] and other papers in the area. As mentioned above, the equivalence relation induces a partition E of the set of states of a complete deterministic FSM. Two states of the same class of the partition E are equivalent; two states of different classes of partition E are distinguishable. Correspondingly, states of the reduced form correspond to classes of the partition E, i.e. the number of states of the reduced form equals the cardinality of E. The transition relation of the reduced form is derived based on transitions between E classes.

For complete deterministic FSMs with time guards [10], the minimization procedure is almost the same. The only difference is that TFSM has to be *well-defined*, i.e., at the first step of the procedure, each two transitions, (s, i, o, s_1, g_1, d), $(s, i, o, s_1, g_2, d) \in h_s$ where g_1 and g_2 can be merged into a single timed guard are replaced by a single transition $(s, i, o, s_1, g_1 \cup g_2, d_1)$. In [10], it is shown that there exists a unique well-defined reduced FSM with timed guards that is equivalent to a given TFSM.

For complete deterministic FSMs with timeouts [11], the minimization procedure is based on the corresponding FSM abstraction and in fact, according to a procedure proposed in [11], a complete deterministic FSM with timeouts can have two non-isomorphic reduced forms. In order to have the minimal form of a complete deterministic FSM with timeouts, the input timeouts should be also minimized and in this paper, we show how this can be done.

We now consider an example where the model of a FSM with timed guards and timeouts can be used. More examples can be found in [13]. In our example, we consider a part of the system for elevator management; a corresponding TFSM is shown in Figure 2 where for the sake of simplicity, some transitions are not shown.

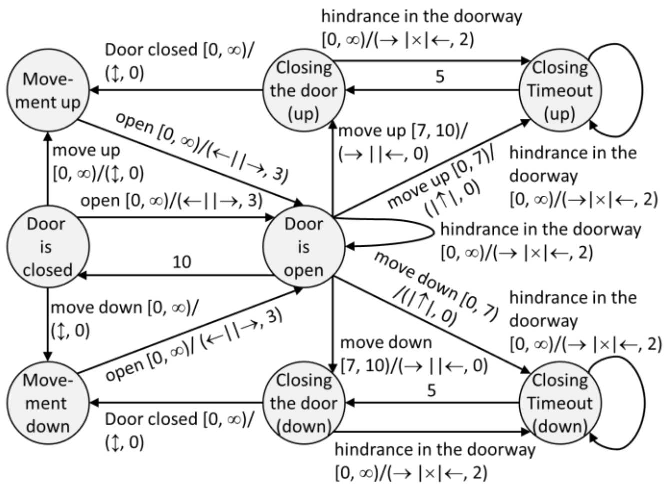

hindrance in the doorway [0, ∞)/(→ |×|←, 2)
Door closed [0, ∞)/ (↕, 0)
Move-ment up
Closing the door (up)
5
Closing Timeout (up)
open [0, ∞)/(←||→, 3)
move up [0, ∞)/(↕, 0)
move up [7, 10)/ (→ ||←, 0)
move up [0, 7)/ (|↑|, 0)
hindrance in the doorway [0, ∞)/(→|×|←, 2)
open [0, ∞)/(←||→, 3)
Door is closed
Door is open
hindrance in the doorway [0, ∞)/(→ |×|←, 2)
10
move down [0, 7) /(|↑|, 0)
hindrance in the doorway [0, ∞)/(→ |×|←, 2)
move down [0, ∞)/ (↕, 0)
open [0, ∞)/(←||→, 3)
move down [7, 10)/(→ ||←, 0)
Move-ment down
Closing the door (down)
5
Closing Timeout (down)
Door closed [0, ∞)/ (↕, 0)
hindrance in the doorway [0, ∞)/(→ |×|←, 2)

Fig. 2. A system for the elevator management

At state "Door is closed", the elevator is waiting for a call and moves to the state "Door is open" by an input "open" with the picture ←| |→ as the output. After opening the door the elevator waits for a command "move up" or "move down" in order to start moving. If an input is applied during the timed guard [0, 7), then the TFSM moves to state "Closing timeout (up)/(down)" with the picture |↑| as the output and waits for passing 5 seconds before closing the door; the clock is reset if there are problems in the doorway. At the state "Closing the door (up)/(down)", the door begins to close and the TFSM moves to state "Movement up/down" by an input "Door closed" obtained from corresponding sensors with the picture ↕ as the output. If an input "Move up" or "Move down" is applied at state "Door is open" during the timed guard [7, 10) when door already started closing, the TFSM moves to state "Closing the door (up)/(down)" immediately. The TFSM remains at the state "Movement up/down" until the next stop. If an input is not applied before the timeout (10 sec) expires at the state "Door is open", then the TFSM moves to state "Door is open".

IV. Deriving the FSM abstraction of TFSM

The behavior of a TFSM can be adequately described using a classical FSM that is called the *FSM abstraction* of the TFSM and is derived similar to [9] but in [9] the authors do not consider output delays.

Given a complete deterministic TFSM $\mathsf{S} = (S, I, O, \lambda_S, \Delta_S, s_0)$, the largest finite boundary of timed guards B_S and maximum output delay D, we derive the FSM abstraction of TFSM S as the FSM $\mathcal{A}_\mathsf{S} = (S_A, I \cup \{\mathtt{I}\}, O_A, \lambda_{AS}, s_0)$ where $S_A = \{(s, 0), (s, (0, 1)), \ldots, (s, (B_\mathsf{S} - 1, B_\mathsf{S})), (s, B_\mathsf{S}), (s, (B_\mathsf{S}, \infty)): s \in S\}$, $O_A = \{(o, 0), (o, 1), \ldots, (o, D): o \in O\} \cup \{\mathtt{I}\}$. The input $\mathtt{I}$ is a special input of the FSM abstraction. Given state (s, t_j), $t_j = 0, \ldots, B_\mathsf{S}$, of FSM $\mathcal{A}_\mathsf{S}$ and input i, a transition $((s, t_j), i, (o, d), (s', 0))$ is a transition of the FSM abstraction $\mathcal{A}_\mathsf{S}$ if and only if there exists a transition $(s, i, o, s', g_i, d) \in \lambda_S$ such that $t_j \in g_i$. Given state (s, g_j), $g_j = (0, 1), \ldots, (B_\mathsf{S} - 1, B_\mathsf{S}), (B_\mathsf{S}, \infty)$, of FSM $\mathcal{A}_\mathsf{S}$ and input i, a transition $((s, g_i), i, (o, d), (s', 0))$ is a transition of $\mathcal{A}_S$ if and only if there exists a transition $(s, i, o, s', g, d) \in \lambda_S$ such that $g_i \subseteq g$. In other words, transitions under input $i \in I$ correspond to timed inputs (i, t) where t is 'hidden' as the second item of states of the FSM abstraction $\mathcal{A}_S$. Transitions under the special input `I` correspond to the clock change between non-integer and integer values, or to a timeout transition between states. Given state s such that $\Delta_S(s) = (s', T)$, transitions $((s, n), \mathtt{I}, \mathtt{I}, (s, (n, n + 1)))$ and $((s, (n - 1, n)), \mathtt{I}, \mathtt{I}, (s, n))$ are in the transition relation λ_{AS} if and only if $n < T$. Transition $((s, (n - 1, n)), \mathtt{I}, \mathtt{I}, (s', 0)) \in \lambda_{AS}$ if and only if $n = T < \infty$. In [9], it is shown that the FSM abstraction of complete and deterministic TFSM S is also complete and deterministic.

A timed input sequence α of TFSM S can be transformed into a corresponding input sequence α_{FSM} of the FSM abstraction $\mathcal{A}_S$. In this case, each timed input (i, t) is replaced by sequence $\mathtt{I}, \mathtt{I}, \ldots, \mathtt{I}, i$, of inputs of the FSM abstraction where the number of inputs `I` equals the number of clock transitions between a non-integer and integer values for the time duration t. At the same time the response of the FSM abstraction to sequence $\mathtt{I}, \mathtt{I}, \ldots, \mathtt{I}, i$ equals $\mathtt{I}, \mathtt{I}, \ldots, \mathtt{I}, (o, d)$, where the number of inputs `I` is the same as for the timed input (i, t) and (o, d) is the response of the TFSM to timed input (i, t). Thus, the output sequence of the FSM abstraction γ_{FSM} can be transformed into corresponding timed output sequence γ by removing all outputs `I`. Using the results of the paper [9] the following statement can be established.

Proposition 1. A timed trace α/γ exists for TFSM S if and only if there exists a trace $\alpha_{FSM}/\gamma_{FSM}$ for the FSM abstraction $\mathcal{A}_S$.

According to Proposition 1, all the trace features of a TFSM are preserved for its FSM abstraction and thus, the state equivalence of a TFSM can be analyzed based on a classical FSM. However, states of this classical FSM abstraction have the information about time properties which should be taken into account when minimizing a TFSM. The following statement establishes necessary and sufficient conditions for two TFSM states to be equivalent.

Proposition 2. States s_1 and s_2 of TFSM S are equivalent if and only if states $(s_1, 0)$ and $(s_2, 0)$ of the FSM abstraction $\mathcal{A}_S$ are equivalent.

In fact, states s_1 and s_2 of TFSM S are equivalent if and only if for each input sequence, the output responses at these states coincide. By Proposition 1, the latter means that for each input sequence, the output responses at states $(s_1, 0)$ and $(s_2, 0)$ of $\mathcal{A}_S$ also coincide.

Thus, the conclusion about the state equivalence of a given TFSM can be drawn based on its FSM abstraction and there are methods for checking the equivalence of states of FSM [1].

V. Minimization procedure

In this section, we propose the minimization procedure for FSMs with timeouts and timed guards. The procedure can be seen as the generalization of two minimization procedures for FSMs which only have either timeouts or timed guards elaborated by the authors [10, 11].

Procedure 1 for deriving a (state) reduced form of a complete deterministic TFSM

Step 1: Derive the FSM abstraction $\mathcal{A}_S$ of TFSM S and the partition E_{FSM} into equivalent states of the abstraction $\mathcal{A}_S$.

Step 2: Derive the partition E into equivalent states of TFSM S as follows: states s_1 and s_2 of TFSM S are in the same class of E if and only if states $(s_1, 0)$ and $(s_2, 0)$ are in the same class of E_{FSM}.

Step 3: Derive a reduced form B of TFSM S. Input and output alphabets of TFSM B coincide with those of TFSM S, states $b_1, b_2, \ldots, b_l$ of TFSM B correspond to classes $B_1, B_2, \ldots, B_l$ of E. For each class B_j of E, choose any state $s_j \in B_j$. There exists a transition $(b_j, i, o, b_k, g, d) \in h_B$ if and only if there exists a transition $(s_j, i, o, s_k, g, d) \in h_s$ such that $s_k \in B_k$. There exists a timeout $\Delta_B(b_j) = (b_k, T)$ in TFSM B if and only if there exists a timeout $\Delta_S(s_j) = (s_k, T)$ such that $s_k \in B_k$.

The following proposition confirms that the above procedure returns a reduced form of the given TFSM.

Proposition 3. Given a deterministic complete TFSM S and TFSM B returned by Procedure 1, B is a complete deterministic TFSM and state b_j of B is equivalent to state s_j of TFSM S if and only if $s_j \in B_j$.

Proof. By Proposition 2, states s_1 and s_2 TFSM S are in the same class of partition E if and only if states $(s_1, 0)$ and $(s_2, 0)$ of the FSM abstraction $\mathcal{A}_S$ are in the same class of partition E_{FSM}. Thus, partition E derived at Step 2 of the procedure is the partition of equivalent states of TFSM S. The TFSM B is complete and deterministic because S is a complete deterministic TFSM and for each state b_j, the set of transition 'repeats' the set of transition at state $s_j \in B_j$.

We now show that states s_j and b_j where b_j corresponds to class B_j of partition E such that $s_j \in B_j$ are equivalent. Each transition from state b_j 'repeats' a transition from state $s_j' \in B_j$ which is equivalent to any state s_j of B_j (Proposition 2). At the same time, if $\Delta_S(s_j') = (s_k, T)$ and $\Delta_B(b_j) = (b_k, T)$, $T < \infty$, then state $s_k \in B_k$ is equivalent to s_k' where transitions from state s_k' 'repeat' those from state b_k. Thus, output responses to any timed input at states s_j and b_j coincide. Since a transition $(b_j, i, o, b_j'', g_j', d)$ of TFSM B corresponds to a transition $(s_j, i, o, s_j'', g_j', d)$ of TFSM S, where $s_j'' \in B_j''$, the same reasoning can be applied for states s_j'' and b_j'', and for all next state pairs. Thus, output responses of TFSMs B and S at states b_j and s_j, respectively, coincide for each timed input sequence. In the same way, we can show that states $s_j \in B_j$ and $s_{kj} \in B_k$, $k \neq j$, are not equivalent.

Thus, the following theorem holds.

Theorem 1. Given a complete deterministic TFSM S, the complete deterministic TFSM B returned by Procedure 1 is a (state) reduced form of TFSM S.

After applying Procedure 1 to TFSM S in Figure 1 the TFSM in Figure 3 is obtained and due to Theorem 1, this TFSM is state reduced and equivalent to S, i.e., it is a (state) reduced form of TFSM S.

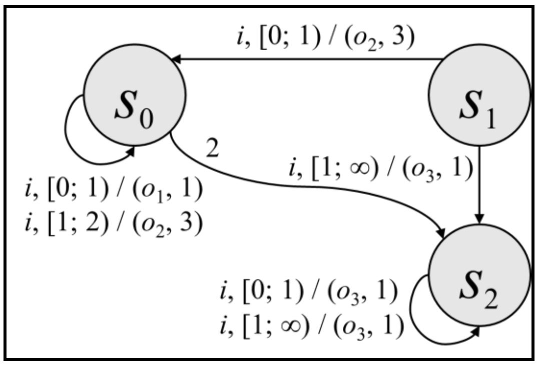


Fig. 3. A reduced form of TFSM S in Figure 1

In general, a state reduced form of a TFSM is not unique, similar to FSMs with timed guards [10] and FSMs with timeouts [11] despite the fact that all reduced forms have the same number of states. The reason is that for timed FSMs not only the state set should be minimized as it happens for classical FSMs but also timed guards and timeouts need to be optimized too. In particular, in some cases, input timed guards at some states could be merged as well as the value of input timeouts could be minimized. For example, timeout $\Delta_S(s_0)$ = (s_2, 2) of TFSM S (Figure 3) can be replaced by timeout $\Delta_S(s_0)$ = (s_1, 1) without breaking the equivalence relation between TFSMs in Figures 1 and 3. Below we introduce a class of well-defined TFSMs for which time aspects are optimized in such a way that any two minimal forms of a TFSM are isomorphic, i.e., the minimal form of a TFSM is unique and can be considered as the canonical representation of a given TFSM.

VI. The uniqueness of TFSM minimal form

A FSM with timed guards and timeouts S is well-defined if for each two transitions (s, i, o, s', g_1, d), (s, i, o, s', g_2, d) $\in h_s$ it holds that timed guards g_1 and g_2 cannot be merged into a single guard. Moreover, for each state s such that $\Delta_S(s)$ = (s', T), it holds that for each state s'' and integer $T' < T$, TFSM S′ which is obtained from S by replacing the timeout at state s to $\Delta_S(s)$ = (s'', T'), is not equivalent to S. Given a deterministic complete TFSM S, a well-defined reduced TFSM P that is equivalent to S is the *minimal form* of TFSM S.

If for two transitions (s, i, o, s', g_1, d), (s, i, o, s', g_2, d) $\in h_s$ of TFSM S timed guards g_1 and g_2 can be merged into a single guard, then these two transitions can be replaced by a transition (s, i, o, s', $g_1 \cup g_2$, d). Minimal timeouts for states of TFSM S can be found based on the FSM abstraction A_S. In order to find a minimal timeout at state s, states (s, 1), …, (s, j), … of the FSM abstraction are considered. If there exists a state (s', 0) such that states (s', 0) and (s, j) are equivalent, then a transition ((s, (j – 1, j)), I, I, (s, j)) of A_S can be replaced by a transition ((s, (j – 1, j)), I, I, (s', 0)). Respectively, the timeout at state s of TFSM S can be replaced by $\Delta_S(s)$ = (s', j) without changing the TFSM behavior. Thus, for each state of the TFSM S, a minimal possible timeout can be chosen that does not change the behavior of TFSM S.

Respectively, given a TFSM S, in order to derive the minimal form of S, i.e., a well-defined equivalent (state) reduced TFSM, two operations can be used: 1) transitions under the same input where timed guards can be merged should be replaced by a single transition; 2) each timeout should be set to minimum value. Below a procedure is proposed for deriving a well-defined TFSM.

Procedure 2 for deriving a well-defined equivalent TFSM for a given TFSM S

Step 1. Replace each two transitions (s, i, o, s', g_1, d), (s, i, o, s', g_2, d) of S such that g_1 and g_2 can be merged into a single timed guard, by a transition (s, i, o, s', $g_1 \cup g_2$, d).

Step 2. Derive the FSM abstraction A_S of TFSM S and the partition E_{FSM} into equivalent states of the abstraction A_S.

Step 3. For each state s of TFSM S where $\Delta_S(s)$ = (s', T), determine whether there exists state (s, j) of A_S, j = 1, …, T – 1, with minimal j such that there exists state (s'', 0) which is equivalent to (s, j). If there exists such pair of states (s, j) and (s'', 0) then replace timeout at state s to $\Delta_S(s)$ = (s'', j).

Step 4. For each state s of TFSM S where $\Delta_S(s)$ = (s', ∞), determine whether there exists state (s'', 0) of A_S such that states (s'', 0) and (s, (j, ∞)) are equivalent. If there exists such a state (s'', 0), then replace timeout at state s by $\Delta_S(s)$ = (s'', j + 1).

Step 5. For each state s of TFSM S where $\Delta_S(s)$ = (s', T) and $T < \infty$, remove a transition (s, i, o, s', g, d) such that $g \cap [0, T) = \varnothing$. If for a transition ($s$, i, o, s', $\langle a, b\rangle$, d) it holds that $\langle a, b\rangle \cap [0, T) \neq \varnothing$ but $\langle a, b\rangle$ is not contained in [0, T), then replace a transition (s, i, o, s', $\langle a, b\rangle$, d) by a transition (s, i, o, s', $\langle a, T)$, d).

Proposition 4. Given a complete deterministic TFSMs S, Procedure 2 returns a well-defined TFSM P that is equivalent to S.

Proof. Step 1 where transitions (s, i, o, s', g_1, d) and (s, i, o, s', g_2, d) are merged as a transition (s, i, o, s', $g_1 \cup g_2$, d), does not change behavior of TFSM S. Consider Step 3 of the procedure for replacing timeouts. Let there exist states s and s'' of TFSM S such that $\Delta_S(s)$ = (s', T), $j < T$, and states (s, j) and (s'', 0) of the FSM abstraction A_S are equivalent. In this case, the pair of states (s, j) and (s'', 0) will be identified at Step 3 of the procedure. By Proposition 2, if states (s, j) and (s'', 0) are equivalent then the output response of TFSM S at state s'' to each timed input sequence coincides with the corresponding response of TFSM S at state s at time instance j. Thus, the replacement of a timeout $\Delta_S(s)$ = (s', T) at state s by $\Delta_S(s)$ = (s'', j) does not change the behavior of TFSM S at state s. Similarly, the replacement of a timeout at Step 4 of the procedure does not change the behavior of TFSM S. At Step 5 of the procedure, a transition (s, i, o, s', g, d) is removed if the input timed guard g and [0, T) are disjoint where $\Delta_S(s)$ = (s'', T), since this transition cannot be executed in the TFSM S. Transition (s, i, o, s', g, d) such that input timed interval g = [a, b) intersects [0, T) but the upper bound of g is beyond the value of timeout T cannot be executed in the TFSM S for the interval [T, b). Thus, removing and bound a transition at Step 5 does not change the behavior of TFSM S.

We now show that TFSM P is well-defined. Assume that for state p_0 of TFSM P the timeout $\Delta_P(p_0)$ = (p_1, T_1) is replaced by a timeout $\Delta_P(p_0)$ = (p_2, T_2) where $T_1 > T_2$, and the behavior of TFSM P at state p_0 is not changed. The latter means that in the FSM abstraction A_P there exists either pair of equivalent states (p_0, T_2) and (p_2, 0) or a pair of equivalent states (p_0, (T_2 − 1, ∞)) and (p_2, 0). The pair of equivalent states (p_0, T_2) and (p_2, 0) does not exist in A_P (Step 3 of the procedure). If there exist equivalent states (p_0, (T_2 − 1, ∞)) and (p_2, 0) in A_P then the output response of TFSM P at state p_2 to each timed input sequence coincides with that of TFSM P at state p_0 starting from timed instance $T > (T_2 - 1)$. In this case, by construction of the FSM abstraction A_P has no state (p_0, T_2), but according to Step 4 of the procedure, the timeout at state p_0 is replaced by $\Delta_P(p_0) = (p_2, T_2)$. Thus, equivalent states (p_0, (T_2 − 1, ∞)) and (p_2, 0) do not exist in A_P.

At the same time, each two transitions (s, i, o, s', g_1, d), (s, i, o, s', g_2, d) ∈ h_s, such that guards g_1 and g_2 can be merged as a single guard, are replaced by a single transition (s, i, o, s', $g_1 \cup g_2$, d) (Step 1 of the procedure).

Theorem 2. Two deterministic complete well-defined reduced TFSMs are equivalent if and only if they are isomorphic.

⇐ Let deterministic complete well-defined reduced TFSMs S and P be equivalent. Since both TFSMs are reduced they have the same number of states. Consider the one-to-one correspondence H: $S \to P$ such that $H(s)$ is a state of TFSM P which is equivalent to state s. We now show that for each pair of states s and $p = H(s)$ such that $\Delta_S(s) = (s', T_s)$ and $\Delta_S(p) = (p', T_p)$, it holds that $T_s = T_p$ and $p' = H(s')$. If $T_p < T_s$ then due to the fact that S and P are equivalent, there exists state s'' which is equivalent to state p'. Since states s and p are also equivalent, the timeout at state s can be replaced by $\Delta_S(s) = (s'', T_s')$ where $T_s' = T_p < T_s$. The latter is not possible as S is well-defined. Since P is well-defined, the same reasoning applies when $T_s < T_p$. Thus, $T_p = T_s$. Since states s and p are equivalent, states $time(s, T_s)$ and $time(p, T_p)$ are also equivalent and respectively, $p' = H(s')$. Similar to [10], since TFSMs S and P are equivalent, there exists a transition (s, i, o, s', g, d) ∈ h_s if and only if there exists a transition ($H(s)$, i, o, $H(s')$, g, d) ∈ h_p. Thus, TFSMs S and P are isomorphic.

⇒ Since isomorphic TFSMs coincide up to state renaming, isomorphic TFSMs are equivalent.

Corollary. Given a deterministic complete TFSM S, two well-defined reduced forms of TFSM S are isomorphic.

For example, the minimal form of TFSM S (Figure 1) is shown in Figure 4.

By direct inspection, one can assure that in Figure 4 the timeout $\Delta_S(s_0) = (s_2, 2)$ has been replaced by the timeout $\Delta_S(s_0) = (s_1, 1)$, while a transition (s_0, i, o_2, s_0, [1, 2), 3) has been removed. At the same time, two transitions from state s_2 have been merged and timeouts $\Delta_S(s_1) = (s_1, \infty)$ and $\Delta_S(s_2) = (s_2, \infty)$ were replaced by $\Delta_S(s_1) = (s_2, 1)$ and $\Delta_S(s_2) = (s_2, 1)$, respectively.

Theorem 3. Given a deterministic complete TFSM S, the minimal form of S is unique.

According to Theorem 3, for FSMs with timed guards and timeouts, there exists the minimal (canonical) form that is a well-defined reduced TFSM.

As a corollary, similar statement can be drawn for FSMs with only timed guards or only with timeouts**.**

Corollary. Given a deterministic complete FSM S with only timed guards (only with timeouts), the minimal form of S is unique.

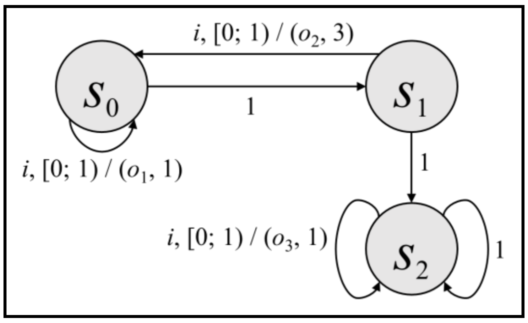
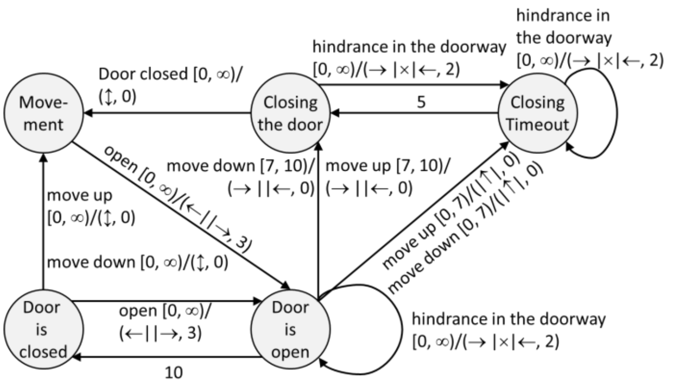


Fig. 4. Well-defined reduced form of TFSM S (Figure 1)

Given the TFSM in Figure 2, by direct inspection, one can assure that there are equivalent states in the TFSM. Namely, state “Closing timeout up” is equivalent to state “Closing timeout down”; state “Closing the door up” is equivalent to state “Closing the door down”, state “Movement up” is equivalent to state “Movement down”. When a TFSM is implemented by the use of a microcontroller tests which check transition and output faults have to be applied to an implementation at hand. However, to the best of our knowledge, the known methods ask for a minimal specification TFSM. Correspondingly, the minimal form of a system in Figure 1 should be derived by Procedures 1 and 2; the result is shown in Figure 5.



hindrance in the doorway [0, ∞)/(→ |×|←, 2)
hindrance in the doorway [0, ∞)/(→ |×|←, 2)
Door closed [0, ∞)/ (↕, 0)
Move-ment
Closing the door
5
Closing Timeout
open [0, ∞)/(←||→, 3)
move down [7, 10)/ (→ ||←, 0)
move up [7, 10)/ (→ ||←, 0)
move up [0, ∞)/(↕, 0)
move up [0, 7)/(|↑|, 0)
move down [0, 7)/(|↑|, 0)
move down [0, ∞)/(↕, 0)
Door is closed
open [0, ∞)/ (←||→, 3)
Door is open
hindrance in the doorway [0, ∞)/(→ |×|←, 2)
10

Fig. 5. The minimal form of the TFSM for the elevator management

VII. Experimental results

Procedures 1 and 2 for deriving the minimal form of a TFSM have been implemented. When experimenting with this program implementation, randomly generated TFSMs with ten inputs and outputs, $|I| = |O| = 10$, and two timed guards in the form $[a, a + 2)$ have been considered. Finite timeouts at states did not exceed four. We generated machines with 10, 20, …., 90, 100 states, 1000 TFSMs for each value. Experimental results are presented in Figure 6.

The performed experiments show that the complexity of the minimization procedure is polynomial with respect to the number of states of a TFSM; moreover, the complexity is almost not increased compared with classical FSMs when the largest finite boundary of timed guards is rather small. However, more experiments are needed to evaluate the efficiency of proposed procedures for TFSMs which describe the behavior of real systems.

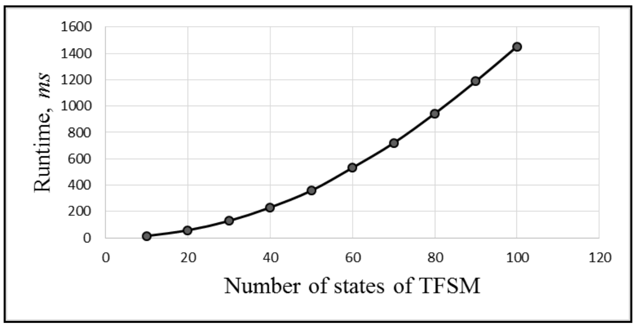


Fig. 6. Runtime dependence on number of TFSM states

VIII. Conclusion

In this paper, a method for deriving the minimal (canonical) form of a Finite State Machine with timed guards and timeouts (TFSMs) has been proposed. We first show how the partition into equivalent states of a given TFSM can be derived based on the corresponding partition of the corresponding FSM abstraction. We also propose the class of well-defined FSMs with timed guards and timeouts and show that given a complete deterministic TFSM, the (state) reduced well-defined equivalent TFSM is unique. A procedure for deriving such a canonical form is proposed and illustrated by examples.

Acknowledgment

This work is partly supported by RSF Project No. 16-49-03012.

References

[1] Gill A. Introduction to the Theory of Finite-State Machines. 1962, 207 p

[2] Lee, D., Yannakakis, M. (1996) Principles and methods of testing finite state machines-a survey. Proceedings of the IEEE. 84(8) (1996) 1090-1123.

[3] T.E. Murphy, X.-J. Geng, and J. Hammer. On the control of asynchronous machines with races. IEEE Transactions on Automatic Control, 48(6):1073-1081, 2003.

[4] R. Kumar, V.K. Garg. Modeling and control of logical discrete event systems. Kluwer Academic Publishers, 1995.

[5] C. C. Cassandras, S. Lafortune. Introduction to discrete event systems. Kluwer Academic Publishers, 1999.

[6] Rita Dorofeeva, Khaled El-Fakih, Stephane Maag, Ana R. Cavalli, Nina Yevtushenko FSM-based conformance testing methods: A survey annotated with experimental evaluation // Information and Software Technology, 2010, 52, pp. 1286-1297.

[7] Maxim Zhigulin, Nina Yevtushenko, Stéphane Maag, Ana R. Cavalli FSM-Based Test Derivation Strategies for Systems with Time-Outs // Proc. of the 11th International Conference on Quality Software, QSIC 2011, IEEE, 2011. pp. 141-149.

[8] M. Merayo, M. Nunez, I. Rodriguez, Formal testing from timed finite state machines, Comput. Netw. 52 (2) (2008) 432–460.

[9] Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. Intern Conf. GANDALF, 2014, pp. 203-216.

[10] A. Tvardovskii, N. Yevtushenko. Minimizing timed Finite State Machines // Tomsk State University Journal of control and computer science, № 4 (29) 2014, pp. 77-83.

[11] Tvardovskii. A. On the minimization of timed Finite State Machines. Proceedings of ISP RAS, Vol. 26, Issue 6, 2014, pp. 77–84.

[12] Khaled El-Fakih, Nina Yevtushenko, Hacène Fouchal. Testing Timed Finite State Machines with Guaranteed Fault Coverage. TestCom/FATES, 2009, pp. 66-80.

[13] O. Kondratyeva. Timed FSM Startegy for Optimizing Web Service Compositions w.r.t. the Quality and Safety Issues. Thèse de doctorat en Informatique, Paris Saclay, 2015.

Experiments On Parallel Composition of Timed Finite State Machines

Alexander Sotnikov[1], Natalia Shabaldina[2], Maxim Gromov[3]
Department of Information Technologies for Studying Discrete Structures,
Tomsk State University,
Tomsk, Russia
[1]sotnikhtc@gmail.com, [2] nataliamailbox@mail.ru, [3]maxim.leo.gromov@gmail.com

***Abstract* — In this paper we continue our work that is devoted to the parallel composition of Timed Finite State Machines (TFSMs). We consider the composition of TFSMs with timeouts and output delays. We held experiments in order to estimate how often parallel composition of nondeterministic TFSMs (with and without timeouts) has infinite sets of output delays. To conduct these experiments we have created two tools: the first one for converting TFSMs into automata (this tool is integrated into BALM-II), the second one for converting the global automaton of the composition into TFSM. As it was suggested in earlier works, we describe the infinite sets of output delays by linear functions, and it is important to know how often these sets of linear functions appear to justify the importance of future investigations of the TFSM parallel compositions (especially for deriving cascade composition). Results of the experiments show significant amount (around 50 %) of TFSMs with infinite number of output delays. We also estimate the size of the global automaton and the composed TFSM. In the experiments we do not consider global automata with the huge number of states (more then 10000).**

Keywords — Timed finite state machines, parallel composition, BALM-II.

I. Introduction

Different systems, for example web-services, telecommunication protocols digital networks etc. are targeted on interaction with each other. To analyze and synthesize such systems one needs an adequate formal model. *The Finite State Machine* (FSM) has proven to be a classical model for description of *input-output reactive discrete event systems* [1]. Here, by "*input-output reactive*" we mean, that every input action is always followed by output reaction, and by "*discrete event*" we mean, that domains for input and output actions are finite (discrete) sets. In this case, when talking about interacting systems, main concept is *a composition* of FSMs. If there are two communicating systems and the behavior of each system is described by an FSM, then their common work can be described by the composition of those FSMs. Under appropriate assumptions [2,3] this composition will also be an FSM. In this work we consider so-called parallel composition [2]. In the parallel composition the interacting systems work asynchronously in the assumption of a slow environment and this is enough to guarantee the composition to be an FSM again. To build an FSM composition BALM-II (Berkeley Automata and Language Manipulation) can be used [3].

For more precise description of a system one should consider time aspects of its behavior as well. For that reason we need some model which would be appropriate for description of *an input-output reactive timed discrete event system*. Probably the most general way to describe timed discrete event system (not necessarily input-output reactive) is *a timed automaton* [6]. In works [4, 5] authors describe web-services, using language BPEL. In [4] authors tell how they translate web services into timed automata in order to verify them. In [5] the authors use more complicated model – so-called *Timed Extended Finite State Machine*. Then they convert a TEFSM into timed automaton. For further analysis, both, [4] and [5], use UPPAAL [16] as an instrument.

Although timed automata are more than enough to describe any input-output reactive timed discrete event system, they are not convenient for us. The reason is that we would like to keep some room for analysis of the composition. Namely, we would like to use the composition as a specification for a test generation. For the best of our knowledge, methods of a test generation with guaranteed fault coverage for (timed) automata are not well developed (frankly speaking we know only one paper [7], which describes such a method). In contrary, test generation methods for FSMs are well-developed and are still developing [8, 9]. And since parallel composition of FSMs guarantees, that the result is FSM again, we would like to consider some kind of *Timed Finite State Machine* as a model, which would presume this property of the parallel composition. One possible timed augmentation of the FSM was mentioned in [5]. But this model is quite complex and it is not clear, how to build parallel composition for it. Another option to introduce timed FSM is Timed FSM with time guards [10]. The theory of this model is highly developed [11], but it lacks an efficient method to build parallel composition as well as the precious model.

And at last, the model we use in this paper, is the *Timed Finite State Machine with output delays and timeouts* (TFSM) [12-15]. This model allows building parallel composition in the same manner as it is done for common FSM. Given two TFSMs we need to compose. First, the corresponding automata should be built [7], then we compose those automata, obtaining so called *the global automaton of the composition*. And then we need to transform the global

This work is supported by the grant for the basic research №16-49-03012 of Russian Scientific Fund.

automaton into TFSM. In [12] very interesting effect of the parallel composition is shown. It turned out that composition of two TFSMs (with constant delays) can have infinite number of output delays for a some transitions and those delays can be described by a finite set of linear functions $\{b + k \cdot t \mid b, k \in \{0\} \cup \mathbb{N}\}$. The main objective of this paper is to investigate how often this effect occurs. This would justify further development of the theory of TFSM with infinite (countable) number of delays.

In some works [4, 5] authors use UPPAAL [16] as an instrument for manipulation with models. Although UPPAAL is very powerful tool of timed systems analysis it does not suit us, because it does not allow to build the composition explicitly. In [14] we compare some tools that can be used for deriving the parallel composition of TFSMs, and explain why we have chosen BALM-II.

BALM-II was designed to build parallel composition of two FSMs. To be able to use this tool for TFSMs we use well-known transformation of TFSM into FSM, and in this work we create a tool for converting TFSM into automaton and integrate it into BALM-II. After deriving two automata for the given two TFSMs we construct a global automaton (using BALM-II). In work [14] we suggest two approaches for getting output delays from the composition of corresponding automata: first deals with BALM-II once again, and the second is based on analyzing of time loops in the automaton. In this work we create tool for converting global automaton into TFSM based on the second approach.

Moreover, in works [14, 15] we consider TFSMs with output delays (without timeouts). In this work we consider TFSMs with output delays and timeouts.

We use implemented tools to hold experiments. Since we describe the infinite sets of output delays by linear functions, it is important to know how often these sets of linear functions appear. The experimental results show significant amount (around 50 %) of TFSMs with infinite number of output delays. We also estimate the size of the global automaton and the composed TFSM. Unfortunately the upper bound of the number of states in the global automaton is exponential due to fact that the automaton determinization is needed for composition. In order to get the results of the experiments in reasonable time, we throw away all examples for which the number of states in the global automaton is too huge (more than 10000 states).

We see the contribution of the paper as three points. First, the algorithm for deriving TFSM from the given global automaton (for the case, when TFSMs have both output delays and timeouts). Second, new tools that allow to derive the binary parallel composition of TFSMs automatically (taking in the mind that composition of two *automata* can be derived using BALM-II). And probably the main point, the experiments have shown, that the theory of TFSMs with linearly-countable output delays is worth to be developed.

The outline of the paper is as follows. In Section II some preliminaries are given. In Section III we describe the structure of the composition that we consider in our work, and how the components communicate with each other. Section IV is devoted to one of the implemented tools which allow to derive TFSM based on the global automaton; we discuss extraction of output delay functions from the global automaton using an example, and propose an algorithm that is lied in the basis of the tool. Section V describes the experiments and experimental results. Section VI concludes the paper.

II. PRELIMINARIES

A finite automaton S is a 5-tuple $(S, X, s_0, F, \lambda_S)$, where S is a finite nonempty set of states with s_0 as the initial state and $F \subseteq S$ as a set of final (accepting) states; X is an alphabet of actions; and $\lambda_S \subseteq S \times X \times S$ is a transition relation. In this work we consider only finite automata, so we will write simply "automaton" (meaning finite automaton). The transition relation defines all possible transitions of the automaton. The language L_S of automaton S is the set of all sequences α in alphabet X, such that in automaton S there is a sequence of transitions (marked by α) from the initial state to some final state. An FSM S is a 5-tuple $(S, I, O, s_0, \lambda_S)$, where S is a finite nonempty set of states with s_0 as the initial state; I and O are input and output alphabets; and $\lambda_S \subseteq S \times I \times O \times S$ is a transition relation. In FSM all states are final.

Let $\mathbb{N}$ be the set of natural numbers. Let $\mathcal{F} = \{b + k \cdot t \mid b, k \in \{0\} \cup \mathbb{N}\}$ – the set of all possible linear functions. TFSM [12] is an FSM with timeouts and output delays $S = (S, I, O, s_0, \lambda_S, \Delta_S, \sigma_S)$, where 5-tuple $(S, I, O, s_0, \lambda_S)$ is underlying FSM, Δ_S: $S \to S \times (\mathbb{N} \cup \{\infty\})$ is a timeout function that determine maximal time of waiting for input symbol, σ_S: $\lambda_S \to (2^{\mathcal{F}} \setminus \varnothing)$ is an output delay function that determine for each transition time delay for producing output symbol (output timeout).

The semantics of Timed FSM is as follows. We describe the behavior of a system that has time aspects: timeouts and output delays. Timeouts describe the situation when the system comes from one state to another not under the input symbol, but in the case when no inputs are applied during some period of time. In practice it's the case of waiting for the password in internet-banking, etc. As for output delays, the meaning of them is that the output symbol is produced for the given input symbol not immediately but after some period of time. For example, a light can change not immediately after a button is pushed but after some time.

We suppose that there is a global clock (timed variable) and this clock is reset to zero when an input symbol (action) is applied, when an output symbol is produced and when the state of the system is changed (for example, in the case of transition under timeout).

III. COMPOSITION OF TIMED FINITE STATE MACHINES

Parallel composition describes a dialog between two components. The structure of the composition is presented in Figure 1.

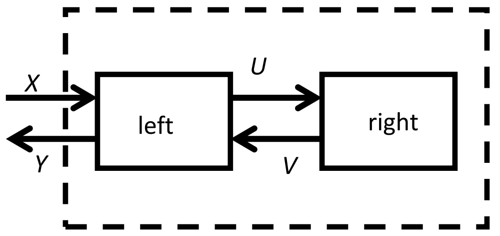


Fig. 1. Structure of binary parallel composition

We suppose that the system works in “slow environment” (it means that the next input can be applied to the composition only after it produces external output to the previous input), the alphabets of different channels don’t intersect and there are no infinite dialogs under internal inputs (it means no livelocks). We also suppose that each component and the whole composition have timed variables. The values of these variables are increasing synchronously, and they reset when the system gets an input or when the state is changed.

In order to compose two TFSMs using BALM-II, we need, first of all, to derive the corresponding automaton for each TFSM [12, 13]. In this work we implement a tool for this step and integrate it into BALM-II as a new command `TFSM2AutV1`. This implementation requires, that MV description of TFSM (BALM-II format) contains special variable called *Time*. Domain of the variable *Time* contains only non-negative integers which are used to describe timeouts and delays. For example, if a table of transitions has head as follows

```
.table I Time O CS -> NS
```

and we would like to represent timeout transition $s_1 \xrightarrow{t} s_2$, then it will appear as

```
^ t ^ s1 s2
```

where *I* – is the variable for input action, *O* – is the variable for output action, *CS* – is the variable for the current state, *NS* – is the variable for the next state, *t* – is some non-negative integer from the domain of the variable *Time* and the symbol ^ represents the fact, that there is no action in corresponding channel. The ordinary transition with delay, like $s_1 \xrightarrow{i/o(d)} s_2$ is described as

```
i d o s1 s2
```

where *i* – is from the domain of *I*, *o* – is from the domain of *O*, and *d* – is from the domain of *Time*.

Then we derive parallel composition of two automatons using BALM-II (we describe how to do this in works [14, 15]). The resulting automaton is so-called global automaton and it describes the common behavior of two automata that are working together in a dialogue mode.

After deriving a global automaton that describes the common behavior of two given TFSMs we need to construct the corresponding TFSM. We also develop a tool for this step and describe the corresponding algorithm in the next section.

IV. Deriving TFSM Based On The Global Automaton. Extracting Output Delays Functions

Let’s consider an example of a global automaton (Figure 2) and describe how to derive the corresponding TFSM (Figure 3). One can see that after Request there can be output Deliver after 3 + 5*t* or 4 + 5*t* tick counts, where *t* is arbitrary non-negative integer number.

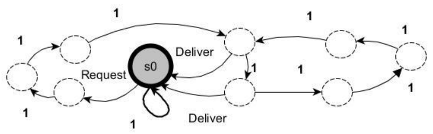


Fig. 2. An example of global automaton

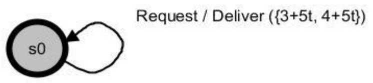


Fig. 3. Corresponding TFSM

In work [14] we propose a procedure for deriving TFSM based on the global automaton for the case when the given TFSMs have only output delays (no timeouts). In this work we propose more common algorithm that works also for the case when the given TFSMs have both output delays and timeouts. The idea of this algorithm is very simple. According to the theory the final states of the global automaton correspond to the states of TFSM. Every sequence, which starts and finishes at some final states of the automaton and goes through non-final states, corresponds to the transition of the TFSM. The sequence can start only with input action or with special action 1. If this sequence starts with special action 1, then every action of the sequence is 1 and corresponding transition is timeout transition (timeout is the number of 1^s needed to reach final state). If the first action of the sequence is input action, then the last action is output action and the intermediate actions are 1^s. In this case the corresponding TFSM transition is ordinary input-output transition with delay. The delay – is the number of 1^s in-between the input and the output actions of the sequence. We just need to keep in mind, that sequence of 1^s may form a loop. So we do some precautions to detect loops when traversing the automaton transitions. The number of 1^s before the loop gives us *b* for linear function and the length of the loop gives *k* for the function.

Algorithm 1. Deriving TFSM based on the global automaton.

Input. Global automaton $A = \langle A, I \cup O \cup \{1\}, a_0, F, \lambda_A \rangle$

Output. TFSM $T = \langle T, I, O, t_0, \lambda_T, \Delta_T, \sigma_T \rangle$ with the same behavior.

```
$t_0 \equiv a_0$; $T := \{t_0\}$;
FOREACH non-visited state $t$ from $T$ DO
  IF $\exists \langle t, 1, t' \rangle \in \lambda_A$ THEN DO
```

```
    ADD t′ in T;
    IF t == t′ THEN ADD ⟨t, ∞, t′⟩ in $\Delta_T$;
    ELSE ADD ⟨t, 1, t′⟩ in $\Delta_T$;
  DONE
  FOREACH input action i such, that
          ∃ ⟨t, i, t′⟩ ∈ $\lambda_A$
  DO
    b := 0; V := ∅;
    WHILE t′ != NULL AND t′ is NOT visited
    DO
      t′.b := b;
      FOREACH output action o such, that
              ∃ ⟨t′, o, t″⟩ ∈ $\lambda_A$
      DO
        IF t″ ∈ F THEN DO
          ADD t″ in T;
          ADD ⟨t′, o, t″⟩ in V;
        DONE
      DONE
      mark t′ as visited;
      b++;
      IF ∃ ⟨t′, 1, t″⟩ ∈ $\lambda_A$ THEN t′ := t″;
      ELSE t′ := NULL;
    DONE
    IF t′ == NULL THEN k := 0, $n_{loop}$ := ∞;
    ELSE k := b – t′.b, $n_{loop}$ := t′.b;
    FOREACH ⟨t′, o, t″⟩ in V DO
      ADD ⟨t, i, o, t″⟩ in $\lambda_T$;
      IF t′.b < $n_{loop}$ THEN
        ADD ⟨⟨t, i, o, t″⟩, t′.b⟩ in $\sigma_T$;
      ELSE
        ADD ⟨⟨t, i, o, t″⟩, t′.b + k*x⟩
                                        in $\sigma_T$;
    DONE
  DONE
  mark t as visited;
DONE
```

V. Experimental Results

We conduct the experiments according to the following steps:

Step 1. Generate two complete nondeterministic observable timed FSMs: `left.fsm` and `right.fsm`. At this step we use FSM generator from the tool [17].

Step 2. Convert generated TFSMs into `AUT`-format (BALM-II format). After this step we have two files in `AUT`-format: `left.aut` and `right.aut`.

The number of all states in the automaton is $S + S*I*D + S*T$, where S – is the number of states in original TFSM, I – number of input symbols, D – maximal delay, T – maximal finite timeout. The number of stable states is $S*T$.

If we have no timeouts then $T = 0$ and the number of states of the automaton is $S + S*I*D$.

Step 3. Convert files `left.aut` and `right.aut` with TFSMs into files `left_aut.aut` and `right_aut.aut` with corresponding automata. In order to do this, we created the tool and integrated it into BALM-II. We described the algorithm for this transformation in the work [14]. In this work we only add in that algorithm the transformation for timeout transitions.

Step 4. Derive the global automaton. We derive the global automaton using the same sequence of BALM-II commands as we described in work [14].

The number of final states in the product automaton is $S1*T1*S2*T2$, where $S1$ is the number of final states in the left component, $S2$ is the number of final states in the right component, $T1$ is the maximal finite timeout in the left component, $T2$ is the maximal finite timeout in the right component.

After the restriction we will have the global automaton with at most $2^{S1*T1*S2*T2} – 1$ states (since the `restriction` command includes determinization of the automaton).

If we have no timeouts then the number of final states in the product automaton is at most $S1*S2$ and after restriction we have at most $2^{S1*S2} – 1$.

Step 5. Derive TFSM based on the global automaton. For this step we created the tool based on the algorithm that was proposed in the previous section.

We generated one hundred pairs of TFSMs for each set of parameters values (number of states, maximal time delays and timeouts). In order to get results of the experiments in reasonable time we fixed the number of inputs and outputs for each channel to 2. We also need to mention that in experiments we did not consider global automata with the huge number of states (more than 10000). It means that we have thrown away such examples. The reason is that the upper bound of the number of states in the global automaton is exponential because of determinization used during composition.

The experimental results are represented in Table 1. In the 3rd and 4th columns there are percentages of TFSMs with infinite number of output delays (we need to use linear functions for describing output delays). The difference is that for the 3rd column we calculated percentage of such TFSMs for the case when the components are TFSMs with timeouts and output delays, and for the 4th column – only output delays (no timeouts). First of all, we would like to comment on dashes (‘-’) in the 3rd and 4th columns. In those cases we could not conduct the experiments for the given parameters in the reasonable time and the reason is the exponential upper bound of the state’s number in the global automaton. For example, let’s consider the last row in the Table 1: we have 4 states in the left TFSM and 4 states in the right TFSM, the maximal finite timeout for the both is the same and it is equal to 7. According to our experiments’ procedure, we first derive the corresponding automata for the given TFSMs. The number of final states in the automata is $S*T$, where S is the number of states in original TFSM, T – maximal finite timeout. So for our case the number of states in the automaton for the left component (let's denote it as $S1$) will be equal to the number of states in the automaton for the right component (let's denote it as $S2$) and $S1 = S2 = = 4*7 = 28$. So, each automaton will have 28 stable states. Then, we estimate the number of states in the product automaton as $S1*T1*S2*T2$, where $T1$ is the maximal finite timeout in the left component, $T2$ is the maximal finite timeout in the right component, so, for our case the number of states in the product automaton will be 28*7*28*7 = 38416. After the restriction we will have the global automaton with at most $2^{S1*T1*S2*T2} – 1$ states since the command `restriction` does determinization of the automaton, and for the last row in our table in the worst case it can be $2^{38416} – 1$ states and of cause it's too huge automaton to deal with.

TABLE I. EXPERIMENT RESULTS

Number of states	***Maximal delay / timeout***	***Percent of TFSMs with infinite number of output delays (with timeouts)***	***Percent of TFSMs with infinite number of output delays (without timeouts)***
2	2	38	23
3	2	39	37
4	2	43	28
5	2	34	-
2	3	47	38
3	3	56	42
4	3	66	55
2	4	46	42
3	4	61	47
4	4	63	53
2	5	67	36
3	5	-	51
4	5	34	69
2	6	-	52
3	6	-	54
4	6	-	68
2	7	-	51
3	7	-	50
4	7	-	75

According to our experimental results, around 50 % of TFSMs, that describe the behavior of the composition, has the infinite number of output delays, so, further investigations of such compositions are needed. It is an actual task especially for the case of cascade composition [15], when each component is a TFSM with timeouts and final sets of output delays, and we first compose two internal components and then we need to compose the resulting TFSM with the remaining part of the system. However, according to our experimental results, this resulting TFSM has infinite number of output delays with high probability. So, more investigations of such compositions are needed.

VI. CONCLUSIONS

This paper is devoted to parallel composition of Timed Finite State Machines (TFSMs). We consider the composition of TFSMs with transitions under timeouts and output delays. It is known that even for the case when output delays are the finite sets of nonnegative integers, the result of such composition can be a TFSM with infinite set of output delays, and we describe such infinite sets by linear functions. It is important to know how often these sets of linear functions appear in order to estimate the importance of future investigations such compositions (especially for deriving cascade composition). In order to conduct the experiments we created two tools: the first one for converting TFSM into automaton (we integrated it into BALM-II), the second one for converting the global automaton into TFSM. The experimental results show significant amount (around 50 %) of TFSMs with infinite number of output delays, so, further investigations of such compositions are needed. We also estimate the size of global automaton and the composed TFSM. In experiments we do not consider global automata with the huge number of states (more then 10000). The reason is that the upper bound of the number of states in the global automaton is exponential because of determinization used during composition. We plan to propose another approach for deriving the composition of Timed Finite State Machines. It will be the part of our future work.

REFERENCES

[1] Gill A. Introduction to the theory of finite state machines, New-York, McGraw-Hill, 1962.

[2] N. Yevtushenko, T. Villa, R. Brayton, A. Petrenko, and A. Sangiovanni-Vincentelli. Solution of parallel language equations for logic synthesis // In The Proceedings of the International Conference on Computer-Aided Design. 2001. P. 103–110.

[3] G. Castagnetti, M. Piccolo, T. Villa, N. Yevtushenko, A. Mishchenko, Robert K. Brayton. Solving Parallel Equations with BALM-II // Technical Report No. UCB/EECS-2012-181, Electrical Engineering and Computer Sciences University of California at Berkeley. 2012. [Electronic resource] http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-181.pdf (date of access: 21.04.2016).

[4] Gregorio Diaz, Juan-Jos e Pardo, Mar a-Emilia Cambronero, Valent n Valero, and Fernando Cuartero. Automatic Translation of WS-CDL Choreographies to Timed Automata, volume 3670 of Lecture Notes in Computer Science, book section 17, pages 230{242. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28701-8. doi: 10.1007/11549970 17. URL http://dx.doi.org/10.1007/11549970_17.

[5] M. Lallali, F. Zaidi, and A. Cavalli. Timed modeling of web services composition for automatic testing. In Signal-Image // Technologies and Internet-Based System, 2007. Bibliography 102 SITIS '07. Third International IEEE Conference on, pages 417{426, Dec 2007. doi: 10.1109/SITIS.2007.110.

[6] R. Alur and D. L. Dill. A theory of timed automata // Theoretical computer science. 1994. Vol.126, Iss. 2. P. 183–235.

[7] Springintveld J., Vaandrager F. and D’Argenio P. Testing timed automata // Theoretical Computer Science, 254 (1-2). pp. 225-257, 2001.

[8] Kushik N., Lopez J., Cavalli A., Yevtushenko N. Improving Protocol Passive Testing through 'Gedanken' Experiments with Finite State Machines // Proceedings - 2016 IEEE International Conference on Software Quality, Reliability and Security, pp. 315-322.

[9] Hierons R., Turker U. Parallel Algorithms for Testing Finite State Machines: Generating UIO Sequences // IEEE Transactions on Software Engineering, 42(11),7429774, pp. 1077-1091, 2016.

[10] K. El-Fakih, M. Gromov, N. Shabaldina, N. Yevtushenko. Distinguishing Experiments for Timed Non-Deterministic Finite State Machines // Acta Cybernetica. 2013. Vol. 21, № 2. P. 205–222.

[11] Tvardovskii A., Yevtushenko N. Minimizing timed Finite State Machines // Vestnik TGU [The Bulletin of TSU], 2014. Vol. 4 (29). P. 132.

[12] O. Kondratyeva, N. Yevtushenko, and A. Cavalli. Parallel composition of nondeterministic finite state machines with timeouts // Journal of Control and Computer Science. Tomsk State University, Russia. 2014. Vol. 2(27). P. 73–81.

[13] O. Kondratyeva, N. Yevtushenko, A. Cavalli. Solving parallel equations for Finite State Machines with Timeouts // Trudy ISP RAN [The Proceedings of ISP RAS]. 2014. Vol. 26, Iss. 6. P. 85–98.

[14] Shabaldina N., Gromov M. Using BALM-II for deriving parallel composition of timed finite state machines with outputs delays and timeouts: work-in-progress // Sistemnaya informatika. 2016. № 8. P. 33-42.

[15] Gromov M. .L, Shabaldina N. V. Using balm-ii for deriving cascade parallel composition of timed finite state machines // Modeling and Analysis of Information Systems, 23:3 (2016), p.699-712.

[16] http://www.uppaal.com/

[17] N. Shabaldina , M. Gromov. FSMTest-1.0: a manual for researches // Proceedings of IEEE East-West Design & Test Symposium (EWDTS’2015). Ukraine, Kharkov: SCITEPRESS, 2015. P. 216-219.

Mining Hybrid UML Models from Event Logs of SOA Systems

Ksenia V. Davydova
National Research University
Higher School of Economics,
PAIS Lab. at the Faculty of Computer Science,
20 Myasnitskaya st.
Moscow, 101000, Russia
Email: kvdavydova@edu.hse.ru

Sergey A. Shershakov
National Research University
Higher School of Economics,
PAIS Lab. at the Faculty of Computer Science,
20 Myasnitskaya st.
Moscow, 101000, Russia
Email: sshershakov@hse.ru

Abstract—**In the paper we consider a method for mining so-called "hybrid" UML models, that refers to software process mining. Models are built from execution traces of information systems with *service-oriented architecture* (SOA), given in form of event logs. While common reverse engineering techniques usually require the source code, which is often unavailable, our approach deals with event logs which are produced by a lot of information systems, and some heuristic parameters. Since an individual type of UML diagrams shows only one perspective of a system's model, we propose to mine a combination of various types of UML diagrams (namely, *sequence* and *activity*), which are considered together with *communication* diagrams. This allows us to increase the expressive power of the individual diagram. Each type of diagram correlates with one of three levels of abstraction (workflow, interaction and operation), which are commonly used while considering web-service interaction. The proposed algorithm consists of four tasks. They include splitting an event log into several parts and building UML sequence, activity and communication diagrams. We also propose to encapsulate some insignificant or low-level implementation details (such as internal service operations) into activity diagrams and connect them with a more general sequence diagram by using *interaction use* semantics. To cope with a problem of immense size of synthesized UML sequence diagrams, we propose an abstraction technique based on regular expressions. We prototype our approach as a Windows-application in C#. It produces UML models in the form of XML-files. The latter are compatible with well-known Sparx Enterprise Architect and can be further visualized and utilized by that tool.**

Index Terms—**Event log, process mining, hybrid UML model, UML sequence diagram, UML activity diagram, reverse engineering.**

I. Introduction

Nowadays we use information systems everywhere. They are used not only at home to increase the comfort of our life but also to support business processes. The complexity of the systems are growing together with the complexity of processes and tasks. Moreover, a lot of systems interact with each other. There is an increasing chance of error as the complexity of the system increases. If the system finds these errors, they are written into so-called event logs together with other information about system execution. The logs store a lot of information during the work of the system. On the one hand, manual processing of the logs is almost impossible because of their size and lack of structure. On the other hand, the event logs are an inestimable source of knowledge about real-life system behavior. Tools, which help to obtain this knowledge in suitable form for analytics are extremely useful.

Different approaches, such as modeling, development within the standardized life cycle, testing, quality assurance (QA), verification, etc., are applied to improve the system quality and error correction. Using combinations of these instruments (for example, testing and verification, modeling and reverse engineering with continuous delivery) gives good results. New tools, modeling tools in particular, help to make the process more convenient and more effective.

Models are built on different life cycle stages. In the classic approach, an architect models an information system based on the customer‘s requirements. However, the implemented system often differs from previously developed models because the system is developed faster than its models. Developers may sometimes make mistakes and may need to spend additional time on critical situations and deadlines. This means that the design and implementation of some components is not completed properly.

When there is no complete model of a system, reverse engineering techniques can be applied to extract the necessary information from the system and build an appropriate model. It allows us to obtain models of a real-life system automatically or semi-automatically. These models correspond to a developed system rather than to an initial plan and initial models. Such models aim both to understand a structure/behavior of a real system and to eliminate any inadequacy of a real model as compared to the initial model. This also makes it easier to fix errors in the system. There are a number of approaches and tools aimed for this purpose. Most of them require the source code of a system to perform analysis. It is not always possible because of different reasons: the source code may not be available to analysts, it is impossible to get the last copy of code or it can be lost. Also, different work groups can develop different system components which complicates centralized collection of source code.

Unlike existing reverse engineering approaches that use source code, we propose an approach that works with system execution traces which can be extracted from event logs. Our approach can be considered as a particular implementation of Process Mining [1], a discipline aimed to discover, analyze and improve business processes and their models. Our approach also includes features that are relevant to software engineering. Hence, we refer to it as *software process mining* [2].

Process mining usually uses process models such as Petri nets, BPMN, Fuzzy maps, etc. which are produced by applying different algorithms such as α-algorithm [1], [3], [4], NLP-algorithm [5] or fuzzy miner [6] respectively. However, these models are not perfectly suitable for software developers. In areas such as software engineering, more specific approaches such as the Unified Modeling Language (UML) [7] are more common. The most common approaches deal with *state class diagrams*, *statecharts*, *sequence* and *activity* diagrams considering them as more descriptive than other. According to UML 2.5, there are two groups of diagrams: structural and behavioral. In this work we primarily focus on the behavioral group, in particular, on *sequence*, *activity* and *communication* diagrams.

Modern approaches to developing information systems make out small reusable well-defined pieces of code, which are commonly refered to as *services*. Systems, using services as a main component, are based on *service-oriented architecture* (SOA) [8]. Services from heterogeneous SOA-systems are developed using different languages, environments and tools, but they work in a single *information space*. Mining unified models of those systems is a challenge and has some difficulties. For example, none of the popular reverse engineering tools works with all languages used for web-service development [9]. As almost all systems produce event logs which contain information about interesting system components, it is possible to build models including all of these components. It simplifies the process of reverse engineering and allows us to expand its application area.

In the paper, we consider event logs written by SOA-systems. Our goal is to expand the applicability of UML-based models for SOA-systems by developing new approaches and tools for mining such models from event logs. UML-standard describes different types of models which suit different modeling aspects of an information system. Nevertheless, there are situations when analysts would like to use expressive opportunities of several diagram types. UML 2.5. does not describe such diagrams, but it does not forbid them. In our paper, we propose a new approach to UML-modeling which includes mining a so-called *hybrid diagram* which includes elements of *UML sequence* and *UML activity diagrams*.

To illustrate the proposed approach, consider the following example.

A. Motivating example

We consider an event log (Table I) produced by an online banking information system with service-oriented architecture. The log contains a number of traces corresponding to individual instances of a business process maintained by the information system. Our goal is to obtain a UML model,

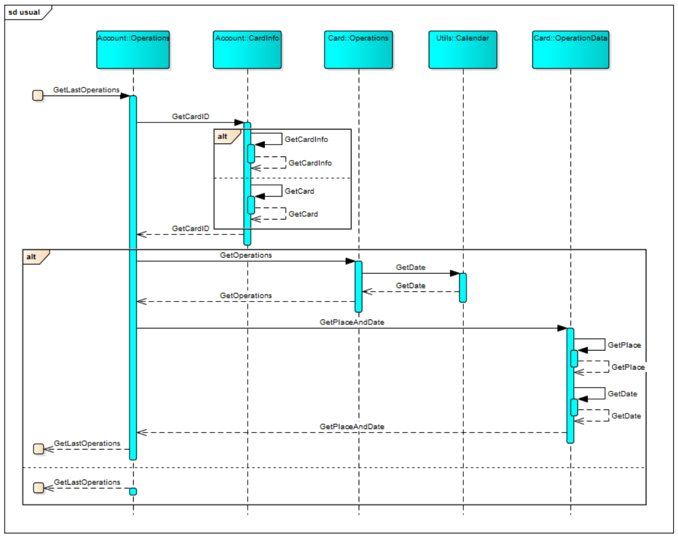


Fig. 1. Usual UML sequence diagram mined from event log L1.

which represents some behavioral aspects of the system from different perspectives [9].

Each row of Table I represents a single event. Columns represent attributes of the log. Events are grouped in cases (by `CaseID` attribute); then, cases are represented in the log by traces. Events are ordered by `Timestamp` attribute. Different components of SOA are represented by other attributes such as `Domain`, `Service/Process` and `Operation`. Domains contain services and processes while the latter consist of operations [10].

By applying a method [9] to the example log, we obtain a UML sequence diagram as depicted in Figure 1 representing the overall process. The diagram contains all possible details (excluding operation parameters) of the behavior of the system as it is represented in the event log. Along with regular messages which connect two different lifelines (depicted as vertical dash lines), the diagram also contains a number of self-calls represented as labeled loop arrows, e.g. `GetCardInfo`, `GetCard`. These self-calls are not important for studying the model from a more abstract perspective. In contrast, they are important when modeling the process of the individual service or another SOA component.

Thus, we propose to hide these calls on the general model with giving a reference to another diagram. Note, that the hidden calls are restricted by one lifeline only. So, using UML sequence diagram here loses its meaning, since only one agent is involved. Therefore, it is convenient to model such behavior by using *UML activity diagram*, another type of UML diagram. Figures 2, 3 and 4 illustrate this idea and represent a *hybrid UML diagram* combining the best features of two different model types.

A distinctive feature of SOA, which is considered, is that processes call other processes and services while services do not call other participants. To demonstrate this feature, it is important to show the interaction between one selected service and its direct services-neighbors which it commu-

TABLE I. Log fragment L1. Banking SOA-system

CaseID	Domain	Service/Process	Operation	Action	Payload	Timestamp
23	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client=Maria, manager=Julia	17:32:15 135
23	Account	CardInfo	GetCardID	REQ	user=a, num=0	17:32:15 250
23	Account	CardInfo	GetCardInfo	REQ	num=0	17:32:15 260
23	Account	CardInfo	GetCardInfo	RES	date=07/16, name=MARIA GRISHINA, id=15674839	17:32:15 267
23	Account	CardInfo	GetCardID	RES	res=15674839	17:32:15 297
23	Card	Operations	GetOperations	REQ	days=30	17:32:15 378
23	Utils	Calendar	GetDate	REQ	days=30	17:32:15 409
23	Utils	Calendar	GetDate	RES	res=23.06.2015	17:32:15 478
23	Card	Operations	GetOperations	RES	res={BP Billing Transfer}	17:32:15 513
23	Card	OperationData	GetPlaceAndDate	REQ	op=BP Billing Transfer	17:32:15 559
23	Card	OperationData	GetPlace	REQ	op=BP Billing Transfer	17:32:15 563
23	Card	OperationData	GetPlace	RES	res=RUS SBERBANK ONLAIN PLATEZH	17:32:15 571
23	Card	OperationData	GetDate	REQ	op=BP Billing Transfer	17:32:15 575
23	Card	OperationData	GetDate	RES	res=20.07.2015	17:32:15 589
23	Card	OperationData	GetPlaceAndDate	RES	res=RUS SBERBANK ONLAIN PLATEZH, date=20.07.2015	17:32:15 601
23	Account	Operations	GetLastOperations	RES	res=succ	17:32:15 822
25	Account	Operations	GetLastOperations	REQ	user=a, today=23.07.2015, client= Maxim, manager=Julia	17:40:18 345
25	Account	CardInfo	GetCardID	REQ	user=a	17:40:18 408
25	Account	CardInfo	GetCard	REQ	num=0	17:40:18 422
25	Account	CardInfo	GetCard	RES	res=no cards	17:40:18 434
25	Account	CardInfo	GetCardID	RES	res=error	17:40:18 489
25	Account	Operations	GetLastOperations	RES	res=no bounded cards	17:40:18 523

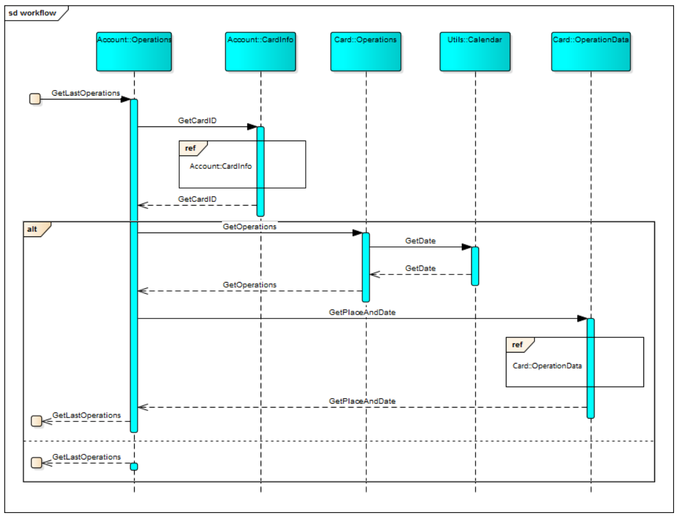


Fig. 2. UML sequence diagram with hidden self calls. High-level diagram of hybrid UML diagram.

nicates with. A UML communication diagram suits this purpose. Example diagrams for `Card::Operations` and `Card::OperationData` processes from example event log are depicted in Figure 5 and 6 respectively. We can see that these processes are called by other processes and call both different services and itself.

We developed a tool which can build hybrid diagrams of UML sequence and activity diagrams automatically. Also, it is able to build a UML communication diagram for a selected SOA component.

B. Related work

Reverse engineering of behavioral UML diagrams is not a new area. There are a number of works [11], [12], [13], [14],

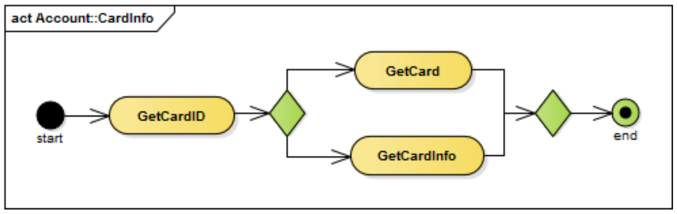


Fig. 3. UML activity diagram with activity inside Account::CardInfo service.

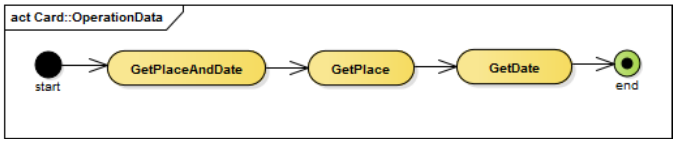


Fig. 4. UML activity diagram with activity inside Card::OperationData service.

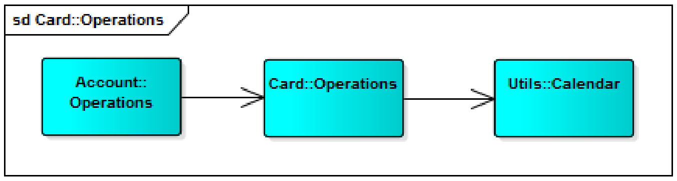


Fig. 5. UML communication diagram for Card::Operations service.

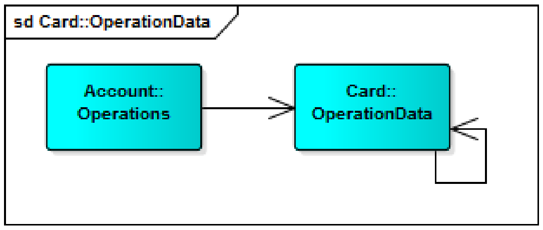


Fig. 6. UML communication diagram for Card::OperationData service.

about building the UML diagrams based on static source code analysis. Besides, there are some CASE tools [15], [16], [17], [18], which can be used for reverse engineering of sequence and activity UML diagrams. There is also a plug-in [19] for development environment NetBeans which is able to build different types of behavioral models from Java source code.

However, all of the above mentioned methods and tools use static program analysis (getting models from source code without execution) for their work. As it was said earlier, source code and all of its versions are not always available for analysis. Thus, these tools and methods are useless in this case. Furthermore, none of these tools are able to infer models from the most popular languages used for developing SOA information systems. Moreover, SOA architecture can be developed with various programming languages. For example, some modules can be written in C#, others can be developed in Java, they can interact with LAMP service, so a single CASE tool cannot produce models for that system. Mining diagrams from event logs solves this problem.

In [20], [21], [22], approaches to building models based on execution traces are proposed. One related work [20] analyzes one trace using meta-models of an event log trace and a UML sequence diagram (UML SD). The trace includes information not only about invocation of methods but also about loops and conditions, which makes easier recognition of fragments such as iteration, alternatives and options. However, program systems logging does not usually include this information, so it is necessary to change the source code to apply this approach.

There is a description of the mining UML sequence diagrams method based on several execution traces in [22]. The authors propose to use a labeled transition system (LTS) as an intermediate model to present one trace and an algorithm to merge LTSs built by several traces. After that, the LTS is transformed into a UML sequence diagram. Moreover, LTS can be used to build a Petri net which can be converted into a UML activity diagram [23]. This conversion possibility can be used to apply different process mining algorithms for receiving a UML activity diagram. The approach to mining hierarchical UML sequence diagrams is proposed in [9] (see Section III-D).

In [24], the authors describe a framework which allows not only behavioral but also static UML diagrams to be built. Their framework generates execution traces by itself from Java source code. After that, the framework is able to build UML activity diagrams from traces but it requires source code for its work.

Process mining proposes to use three abstraction levels for mining models for web services interaction [25]: workflow, interaction and operation. On the operation level only one service is considered in order to look at its internal behavior and functionality. On the interaction level they consider not only one selected service but also its direct callers and callees. Finally, the overall services interaction is covered on the workflow level. We apply all of these levels to service-oriented architecture in the paper.

Furthermore, research on service mining was described in [26]. The author builds different Petri nets for different services (considered at the operation level) and then combines them by places. Thus, he builds a generalized model which refers to the workflow level.

The rest of the paper is organized as follows. Section II gives definitions. Section III introduces our approach to mining hybrid UML models. Section IV contains a description of tool implementation. Section V concludes the paper and gives directions for further research.

II. Preliminaries

$\mathcal{P}(X)$ is the powerset over some set X; Λ is a set of all possible string labels.

Definition 1. (Event log) Let $e = (a_1, a_2, ..., a_n)$ be an event, where a_i is an i-th attribute and n is a number of them. E is a set of events. $\sigma =< e_1, e_2, ...e_k >$ is an event trace where $e_1, ..., e_k$ is an ordered set of events. $Log = \mathcal{P}(E)$ is an event log which is a powerset of traces.

Definition 2. (UML Sequence Diagram) A UML sequence diagram is a tuple $U_{SD} = (L, T, A, P, M, Ref, F)$, where:

- T is a set of moments of discrete time, which determine a partial order over diagram components.
- L is a set of named lifelines. $L = \{l = (\lambda, t) | \lambda \in \Lambda, t \in T\}$
- A is a set of *activations* mapped onto *lifelines*. $a \in A : a = (l, t_b, t_e)$, where $l \in L, t_b, t_e \in T, t_b < t_e$
- $P \subset \Lambda$ is a set of message parameters.
- Ref is a set of *interaction use* (*ref* fragments) which group lifelines and hide them interaction. $ref \in Ref : ref = (L', \lambda)$, where $L' \subset L, \lambda \in \Lambda$
- M is a set of *messages*. $m \in M : m = (a_1, t, \lambda, a_2, type)$, where $a_1, a_2 \in A \cup Ref \cup L, t \in T, \lambda \in P, type \in \{call, return\}$
$a_1 = (l_1, t_{11}, t_{12}), a_2 = (l_2, t_{21}, t_{22}) : t_{11} < t_{21}, t_{11} < t_{12}, t_{21} < t_{22}$
- F is a set of *combined fragments* of the diagram. $F = \{(frag, M') | M' \subseteq M, frag \in \{alt, loop, opt, par\}\}$

Figure 1 represents an example of UML sequence diagram. A *lifeline* is represented as a vertical dashed line with its name at the top. An *activation* is represented as a rectangle on a lifeline, which takes and emits messages (represented as arrows). Message can be *call* and *return* and they contain text parameters. Messages inside one fragment are ordered by time. *Fragments* contain a number of messages and can contain other combined fragments. They are able to show alternatives, loops, parallelisms and other control structures. Another type of fragment, *ref* fragments, refer to other diagrams. Such diagrams can be both UML sequence diagrams and UML activity ones.

Definition 3. (UML Activity Diagram) A UML activity diagram is a tuple $U_{AD} = (N, E, NT)$, where:

- NT is a set of node types. $NT = \{control, object, executable\}$
- N is a set of nodes. $n \in N : n = (\lambda, type)$, where $\lambda \in \Lambda, type \in NT$
- E is a set of edges. $e \in E : e = (n_1, n_2)$, where $n_1, n_2 \in N$

Figure 3 represents an example of a UML activity diagram for `Account::CardInfo` service. Different node types have different meanings. Control nodes represent different behavioral elements such as start, fork and decision. Object nodes represent data (input and output) of an action. Executable nodes represent steps (actions) of the modeling activity. There are three named executable nodes and four control nodes (start, end, decision and merge) in Figure 3. Different control nodes can impose limitations. For instance, start nodes cannot have incoming edges, end nodes cannot have outgoing edges, decision and fork nodes can have only one incoming edge but several outgoing ones; the opposite is true for merge and join.

$\mathfrak{U}_{AD}$ is a set of all possible UML activity diagrams U_{AD}.

Definition 4. (Hybrid UML Diagram) A hybrid UML diagram is a tuple $U_{HD} = (U_{SD}, AD, f)$, where:

- $U_{SD} = (L, T, A, P, M, Ref, F)$ is a UML sequence diagrams.
- $AD \subset \mathfrak{U}_{AD}$.
- $f : Ref \to AD$ is a function which maps *ref* fragments from a UML sequence diagram onto corresponding activity diagram.

Figures 2, 3 and 4 illustrate an example of a hybrid UML diagram. Figure 2 is a UML sequence diagram and represents a high-level diagram. It refers to UML activity diagrams (Figures 3 and 4) using *ref* fragments.

Definition 5. (UML Communication Diagram) A UML communication diagram is a tuple $U_{CD} = (L_{CD}, M_{CD})$, where:

- $L_{CD} \subset \Lambda$ is a set of named lifelines which represent interaction participants.
- M_{CD} is a set of messages. $m_{CD} \in M_{CD} : m_{CD} = (l_1, l_2, \lambda)$, where $l_1, l_2 \in L_{CD}, \lambda \in \Lambda$

Figures 5 and 6 provide examples of UML communication diagrams for two different services.

$\mathfrak{U}_{CD}$ is a set of all possible UML communication diagrams U_{CD}.

Definition 6. (Hybrid UML Model) A hybrid UML model is a tuple $U_{HM} = (U_{HD}, CD)$, where:

- U_{HD} is a hybrid UML diagram.
- $CD \subset \mathfrak{U}_{CD}$.

Figures 2, 3, 4, 5 and 6 represent a hybrid UML model built for example event log L1.

III. Mining Hybrid UML Models

The authors in [25] propose definitions of three levels of abstraction: *operation*, *interaction* and *workflow*. The levels are used for consideration of web service interaction. It motivated us to use different types of UML diagrams which demonstrate features of these levels. In the following sections we consider which UML diagrams suit each abstraction level and why.

Algorithm 1: Building a hybrid UML model U_{HM}

```
Input : an event log $Log$;
an attribute name with REQ/RES value $A_{RR}$;
a set of attributes for mapping onto lifelines $A_L$;
a set of attributes for mapping onto message parameters $A_M$;
a case ID which defines trace for which it is necessary to build model $caseId$;
a set of regular expressions for merging diagram components $L_{RE}$ ;
Output: $U_{HM} = (U_{HD}, CD)$ — hybrid UML model;
begin
    /* Split event log into several parts */
    $Log_w, Log_o \leftarrow$ splitEventLog($Log, A_L, A_{RR}$);
    /* Build activity diagrams using $\alpha$-algorithm [3] */
    $AD \leftarrow$ buildADsAlpha($Log_o$);
    $U_{SD} \leftarrow$ buildSD($Log_w, AD, L_{RE}, A_L, A_M, A_{RR}, caseId$);
    $CD \leftarrow$ buildCDs($Log_w, A_L, ARR$);
    return $U_{HM}$;
```

A. Operation and workflow abstraction levels

Operation level of abstraction shows what is happening inside one isolated service. Activity outside the service is not considered on the operation level. Service is the only process participant. Using a UML sequence diagram leads to a large number of self-calls and *"snowball models"*. It makes the diagram less readable and less understandable. A UML activity diagram suits this purpose since it allows us to demonstrate the complex relationships between operations inside a single participant. Figure 3 shows an example of a UML activity diagram for service `Card::OperationData`.

A business process, provided by services, is represented on a *workflow* abstraction level. There are a lot of participants, so it is useful to use a UML sequence diagram for this level. The diagram is suitable to present not only a sequence of business process actions but also participants of this process and their interaction. An example for event log L1 is depicted in Figure 1.

To bind different abstraction levels, it is necessary to connect them. Our proposal is to use *hybrid UML diagrams* to represent and connect *operation* and *workflow* abstraction levels together. A UML sequence diagram is used to represent a business process at a workflow abstraction level. The diagram contains special objects, *ref* fragments, which make a connection to corresponding UML activity diagram. Every such activity diagram models the behavior of a single service. An example of considered hybrid diagram is presented in Figures 2, 3 and 4.

B. Interaction abstraction level

This level shows interaction of one selected service or process with its nearest neighbors. For a given service its

Algorithm 2: Splitting of an event log into several parts `splitEventLog`

```
Input  : an event log $Log$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $Log_w$ — a part of an event log which contains
         interaction between different services;
$Log_o$ — a set of event logs (parts of initial event log).
Each of them contains events related to an individual
service;
Data:  $f : K \to \mathcal{P}(V)$, where $K$ is a set of keys and
       $\mathcal{P}(V)$ is a set of value sets;
begin
    /* Get lifeline names from an event log */
    $K \leftarrow$ getLifelineNames($Log, A_L$);
    for $\sigma \in Log$ do
        $\sigma' \leftarrow \emptyset$;
        /* stack - stack with nested events */
        $stack \leftarrow \emptyset$;
        for $i \leftarrow 1$ to $|\sigma|$ do
            $e \leftarrow \sigma[i]$;
            $f$(getLifelineName($e, A_L$)) $\leftarrow$
            $f$(getLifelineName($e, A_L$)) $\bigcup\{e\}$;
            if $isRequest(e, A_{RR}) = true$ then
                $e_{prev} \leftarrow stack.peek()$;
                if
                $i = 0 \bigvee getLifelineName(e, A_L)! =$
                $getLifelineName(e_{prev}, A_L)$ then
                    $\sigma' \leftarrow \sigma' \bigcup\{e\}$;
                $stack.push(e)$;
            if $isRequest(e, A_{RR}) = false$ then
                $stack.pop()$;
        $Log_w \leftarrow Log_w \bigcup\{\sigma'\}$;
    for $k \in K$ do
        $Log_o \leftarrow Log_o \bigcup\{f(k)\}$;
    return $Log_w, Log_o$;
```

nearest neighbors are caller and callee services. A UML sequence diagram does not fully suit for representing this level as well as an activity diagram. In the former case, a UML sequence diagram contains a time perspective on which no relation can be mapped. Thus, this leads us to have a "blind" diagram. In the latter case it does not support multiple participants which is important for this abstraction level.

We propose to use UML communication diagrams for depicting processes occurring in SOA system on interaction abstraction level. An example of such a diagram for `Card::Operations` and `Card::OperationData` from an example event log is presented in Figures 5 and 6.

Algorithm 3: Building a UML sequence diagram `buildSD`

```
Input  : an event log $Log$; a set of UML activity
         diagrams $AD$ which $U_{SD}$ will be refer to;
a set of regular expressions for merging diagram
components $L_{RE}$ a set of attributes for mapping onto
lifelines $A_L$; a set of attributes for mapping onto
message parameters $A_M$; an attribute name with
REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Output: $U_{SD} = (L, T, A, \Lambda, M, Ref, F)$ — UML
         sequence diagram referring to UML activity
         diagrams;
begin
    /* Get lifelines from event log */
    $L \leftarrow$ mapLifelines($Log, A_L$) ;
    if $caseId = \emptyset$ then
        $isAlt \leftarrow true$;
        $caseId \leftarrow$
        getCaseIdOfLongestTrace($Log$);
    else
        $isAlt \leftarrow false$;
    /* Get trace with case ID which is equal to $caseId$ */
    $\sigma \leftarrow Log[caseId]$;
    for $i \leftarrow 1$ to $|\sigma|$ do
        $e \leftarrow \sigma[i]$;
        while $isRequest(e, A_{RR}) = true$ do
            if $isAlt = true$ then
                /* Look for differences
                between corresponding events
                in other traces, add found
                events to diagram using
                combined fragments */
                findFrames($Log, caseId, e, U_{SD}, A_M, A_{RR}$);
            else
                /* Get a message parameter
                and add its message to
                diagram */
                mapMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
        while $isRequest(e, A_{RR}) = false$ do
            if $isAlt = true$ then
                findFrames($Log, caseId, e, U_{SD}$);
            else
                mapResponseMessage($e, A_M, M, A, Ref$);
            $i \leftarrow i + 1$;
    if $L_{RE}! = \emptyset$ then
        /* Merge components of the diagram
        using regular expressions */
        changeDiagramUsingREs($U_{SD}, L_{RE}$)
    return $U_{SD}$;
```

C. Building process

Figure 7 represents the workflow diagram of a hybrid mining process. The scheme contains the following tasks (see Algorithm 1):

- An event log is split into several parts. The workflow part of the log refers to services communication. Such communication is represented on a UML sequence diagram at workflow level. The operation parts consist of events referred to activity only inside a specific service.
- A UML sequence diagram is built from a workflow part of an event log using the method proposed in [9] (see Section III-D) extended by a number of necessary *ref* fragments used for connecting with corresponding activity diagrams.
- UML activity diagrams are built from operation log parts independently using one of the process mining algorithms which produces a Petri net. For instance, α-algorithm [4] or inductive miner [27] can be considered here. Then, Petri nets are converted into activity diagrams by a simple translation routine. This conversion is rather trivial since UML activity diagrams are initially based on Petri nets [7], [23].
- Finally, UML communication diagrams are built using the initial event log.

D. Mining UML sequence diagrams

To mine a UML sequence diagram we use a method proposed in [9]. There, we propose an approach to mining UML sequence diagrams with different levels of abstraction. It consists of three steps. The first step of the approach is mapping event log attributes onto UML sequence diagram components. There are two functions for mapping attributes onto lifelines and message parameters. The smaller the SOA element we choose for mapping onto lifelines, the lower the abstraction level we receive.

The second step is set to build a smaller model by applying regular expressions for merging similar messages and lifelines on a diagram. For example, we have two messages with the following parameters: `GetPlaseAndDate, op=BP Billing Transfer` and `GetPlaseAndDate, op=Retail`. They differ in `op` value, thus, these messages can be combined into one message with the following parameter: `GetPlaseAndDate, op=.*`. After the merging, a derived model becomes more generalized and its size decreases in width and height.

To demonstrate the hierarchy of calls, which is important for SOA, a hierarchical diagram can be applied. Thus, the third step of our approach contains a way to present a complex model by using hierarchical UML diagrams. UML standard [7] allows us to divide the model into some parts and connect them by means of *interaction use* (*ref* fragment) and *gates*.

IV. Tool overview

This section presents a brief overview of the software tool implementing the proposed algorithm.

Algorithm 4: Looking for differences between corresponding events in other traces `findFrames`

```
Input : an event log $Log$;
a current event $e$;
a UML sequence diagram
$U_{SD} = (L, T, A, \Lambda, M, Ref, F)$;
a set of attributes for mapping onto message parameters
$A_M$;
an attribute name with REQ/RES value $A_{RR}$;
a case ID which defines trace for which it is necessary to
build model $caseId$;
Data: $Tree$ is a tree with interaction operands
begin
    $equalCases \leftarrow \emptyset$;
    /* Look for corresponding not equal
    events in other traces, group case
    IDs with equal events into $equalCases$
    */
    $notEqEvents \leftarrow$
    findNotEqEvents($e, Log, caseId, equalCases$);
    if $notEqEvents! = \emptyset \bigvee$
    isLastTrace($e, Log) = true$ then
        /* Look for operand where it is
        necessary to add events              */
        $toAdd \leftarrow$ findOperand($equalCases, Tree$);
        addMessagesToFragment($e, equalCases,$
        $toAdd, Tree$);
    else
        if $Tree = \emptyset$ then
            $Tree \leftarrow$ newNode($equalCases$);
        if isRequest($e, A_{RR}) = true$ then
            mapMessage($e, A_M, M, A, Ref$);
        else
            mapResponseMessage($e, A_M, M, A, Ref$);
```

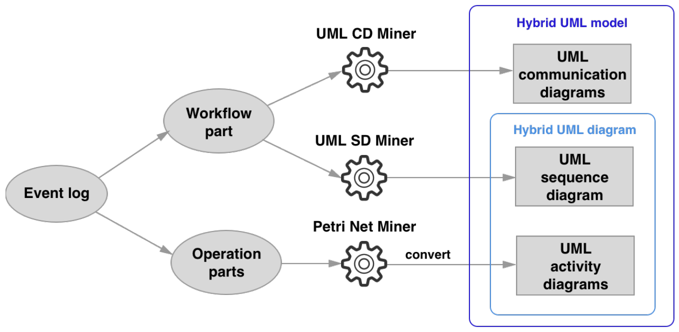


Fig. 7. The workflow diagram of a hybrid mining process.

Algorithm 5: Building UML communication diagrams for each service `buildCDs`

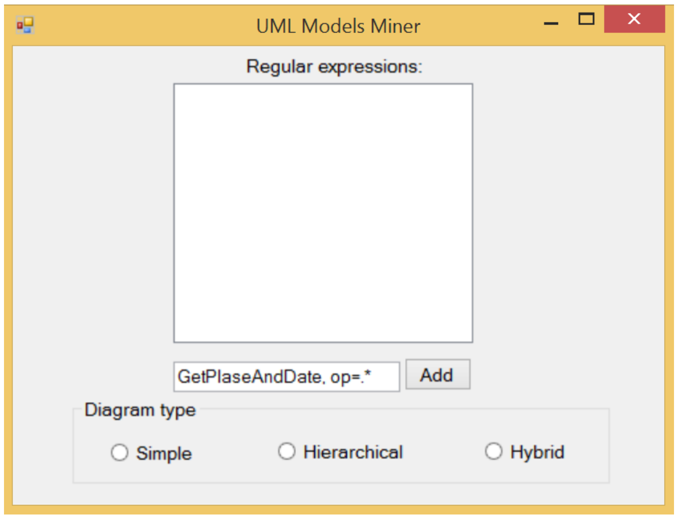
```
Input  : an event log $Log_w$;
a set of attributes for mapping onto lifelines $A_L$;
an attribute name with REQ/RES value $A_{RR}$;
Output: $CD$ — a set of UML communication diagrams
        for each service;

begin
    /* Iterate through lifeline names
    (participants) from an event log    */
    for $l \in getLifelines(Log_w, A_L)$ do
        $L_{CD} \leftarrow \{l\}$;
        $M_{CD} \leftarrow \emptyset$;
        for $\sigma \in Log$ do
            for $i \leftarrow 1$ to $|\sigma|$ do
                $e \leftarrow \sigma[i]$;
                if $i! = 0 \bigwedge getLifeline(e, A_L) = l$
                then
                    $l' \leftarrow \mathtt{getLifeline}(e_{prev}, A_L)$;
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l', l, \emptyset)\}$;
                if $i! = 0 \bigwedge$
                $getLifeline(e_{prev}, A_L) = l$ then
                    $l' \leftarrow \mathtt{getLifeline}(e, A_L)$;
                    if $l' \notin L_{CD}$ then
                        $L_{CD} \leftarrow L_{CD} \bigcup \{l'\}$;
                        $M_{CD} \leftarrow M_{CD} \bigcup \{(l, l', \emptyset)\}$;
                if $isRequest(e, A_{RR}) = true$ then
                    $e_{prev} \leftarrow e$
        $U_{CD} \leftarrow (L_{CD}, M_{CD}, \Lambda)$;
        $CD \leftarrow CD \bigcup \{U_{CD}\}$;
    return $CD$;
```

A. Event log

The tool requires an input event log to be presented in definite format. We use simple CSV text files to represent event logs. An event log should contain some fields that are mapped onto mandatory attributes, namely *CaseID*, *Timestamp* and *Activity*.

B. Tool implementation

The tool is implemented as a Windows application written in C# programming language. The tool allows users to configure main parameters such as regular expressions, hierarchy and type of output diagram (regular UML, hierarchical or hybrid). Regular expressions are applied for merging diagram components. It is implemented as shown in Figure 8. The GUI allows the user to set the type of diagram. The *perspective* of the diagram (a mapping attributes onto diagram lifelines and messages) is set as it described in [9].



GetPlaseAndDate, op=.*

Fig. 8. GUI to set a type of the diagram and regular expressions for merging its components.

The output of the tool is an XMI-file containing a model and a description of diagrams. It can be visualized by Sparx Enterprise Architect [15].

V. Conclusion

This paper introduced a new concept of hybrid UML models and proposed a method of mining them from event logs of SOA information systems using a service mining approach. Our method can also be applied to other types of UML diagrams. The paper discussed approaches to mining diagrams on different abstraction levels.

Our method builds models by using only event logs. This is an advantage over some reverse engineering techniques because the source code is not always available. The proposed method includes mining hybrid UML diagrams which represent workflow abstraction level on UML sequence diagrams and operation level on UML activity diagrams. Moreover, we proposed to build UML communication diagrams to show interaction abstraction level with regards to the service mining approach.

Generally, control structures in system‘s behavior lead to a presence of a big number of nested combined fragments within a UML sequence diagram. It makes the diagram less readable and less understandable. Although UML activity diagrams have no time perspective in contradistinction to sequence diagrams, the former show alternatives, loops and parallelism more clearly. Since there are also a lot of event logs which are not produced by SOA systems, we are going to expand our approach to mining hybrid UML diagrams from event logs of more broad types of software architecture in the future.

Acknowledgement

This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

References

[1] W. M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[2] V. Rubin, C. W. Günther, W. M. P. van der Aalst, E. Kindler, B. F. van Dongen, and W. Schäfer. *Process Mining Framework for Software Processes*, pages 169–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[3] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, and A. J. M. M. Weijters. Process mining: Extending the α-algorithm to mine short loops. In *Eindhoven University of Technology, Eindhoven*, 2004.

[4] W. M. P. van der Aalst, A. J. M. M. Weijter, and L. Maruster. Workflow Mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:2004, 2003.

[5] F. Friedrich, J. Mendling, and F. Puhlmann. *Process Model Generation from Natural Language Text*, pages 482–496. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[6] C. W. Günther and W. M. P. van der Aalst. *Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics*, pages 328–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[7] OMG. OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5, August 2015.

[8] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[9] K. V. Davydova and S. A. Shershakov. Mining Hierarchical UML Sequence Diagrams from Event Logs of SOA systems while Balancing between Abstracted and Detailed Models. 28(3):85–102, 2016.

[10] S. A. Shershakov and V. A. Rubin. System runs analysis with process mining. In *Modeling and Analysis of Information Systems*, pages 818–833, 2015.

[11] A. Rountev and B. H. Connell. Object Naming Analysis for Reverse-engineered Sequence Diagrams. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 254–263, New York, NY, USA, 2005. ACM.

[12] A. Rountev, O. Volgin, and M. Reddoch. Static Control-flow Analysis for Reverse Engineering of UML Sequence Diagrams. *SIGSOFT Softw. Eng. Notes*, 31(1):96–102, September 2005.

[13] P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In *International Conference on Software Maintenance*, pages 159–168. IEEE Computer Society, 2003.

[14] E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In *WCRE*, pages 297–298. IEEE Computer Society, 2006.

[15] Sparx Systems' Enterprise Architect. http://www.sparxsystems.com.au/products/ea/.

[16] IBM Rational Software Architect. https://www.ibm.com/developerworks/downloads/r/architect/.

[17] Visual Paradigm. https://www.visual-paradigm.com/features/.

[18] Altova UModel. http://www.altova.com/umodel.html.

[19] NetBeans UML. http://plugins.netbeans.org/plugin/1801/netbeans-uml.

[20] L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. *IEEE Trans. Softw. Eng.*, 32(9):642–663, September 2006.

[21] R. Delamare, B. Baudry, and Y. Le Traon. Reverse-engineering of UML 2.0 Sequence Diagrams from Execution Traces. In *Proceedings of the workshop on Object-Oriented Reengineering at ECOOP 06*, Nantes, France, July 2006.

[22] T. Ziadi, M. A. A. da Silva, L. M. Hillah, and M. Ziane. A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams. In Isabelle Perseil, Karin Breitman, and Roy Sterritt, editors, *ICECCS*, pages 107–116. IEEE Computer Society, 2011.

[23] B. Agarwal. Transformation of UML Activity Diagrams into Petri Nets for Verification Purposes. 2(3):798–805, 2013.

[24] A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer. fREX: FUML-based Reverse Engineering of Executable Behavior for Software Dynamic Analysis. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering*, MiSE '16, pages 20–26, New York, NY, USA, 2016. ACM.

[25] S. Dustdar, R. Gombotz, and K. Baina. Web Services Interaction Mining. Tech. Rep. TUV-1841-2004-16. 2004.

[26] W. M. P. van der Aalst. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. *IEEE Transactions on Services Computing*, 6(4):525–535, 2013.

[27] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst. *Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour*, pages 66–78. Springer International Publishing, Cham, 2014.

Tool for Behavioral Analysis of Well-Structured Transition Systems

Leonid Dworzanski
Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
leo@mathtech.ru

Vladimir Mikhaylov
Department of Software Engineering
National Research University Higher School of Economics
Moscow, Russia
vemikhaylov@edu.hse.ru

***Abstract*— Well structured transition systems (WSTS) became a well known tool in the study of concurrency systems for proving decidability of properties based on coverability and boundedness. Each year brings new formalisms proven to be WSTS systems [11,12]. Despite the large body of theoretical work on the WSTS theory, there has been a notable gap of empirical research of well-structured transition systems. In this paper, the tool for behavioural analysis of such systems is presented. The WSTS systems are described via the extension of SETL language. It makes the description of the formalism close to the formal definition. It allows to easily introduce new formalisms and conduct analysis of the behavioural properties without programming efforts. Two most studied algorithms for analysis of well-structured transition systems behavior (backward reachability and the Finite Reachability Tree analyses) have been implemented; and, their performance was measured through the runs on such models as Petri Nets and Lossy Channel Systems. The developed tool can be useful for analysis of different subclasses of well-structured transition systems.**

Keywords—formal verification; infinite systems; well-quasi-ordering; Petri net

I. Introduction

Formal verification provides researchers and developers with approaches that are widely-used for proving that a program satisfies a formal specification of its behavior. These methods are highly demanded in the software and hardware engineering, as they provide appropriate level of systems reliability; which, in most cases, cannot be ensured by simulation.

One of the most common technique of formal verification is model checking or property checking. It involves algorithmic methods that are applied to check satisfiability of a logic formula used for the representation of the model of the system and the specification. The main advantage of model checking is considered to be the fact that it enables almost completely automatic process of verification. Model checking proved to be effective in practice for analysis of finite-state systems [1]; however, in case of systems with infinite state space the situation is more complicated because exhaustive search, which is usually used by verification tools, cannot be applied directly.

In order to deal with infinite-state systems Finkel proposed the idea of well-structured transition systems (WSTS) in 1987 [2]. “These are transition systems where the existence of a well-quasi-ordering over the infinite set of states ensures the termination of several algorithmic methods. [3]” The suggested model has provided researchers with an abstract generalization of several models (i.e. Petri nets, lossy channel systems and timed automata). Therefore, the results obtained from the analysis of such a generalized model can be also applied to these specific models.

The WSTS analysis can be used to solve, for instance, covering, termination, inevitability and boundedness problems. However, the application of the WSTS analysis is hampered by the necessity of implementing algorithms and data structures to support the analysis for each new formalism. In this work, the tool that can be used for analysis of WSTS is presented. We introduce the WSTSL language - modification of SETL language – set-theoretical programming language. The language provides the user with opportunity to define the structure of analyzed system very close to the original formal definition. After definition of the formalism, it is immediately possible to run backward reachability method [4] or the Finite Reachability Tree [5] on it. It allows a computer scientist to conduct the analysis of a WSTS system almost instantly after proving the well-structuredness of his or her formalism, and to postpone the implementation phase after what-if experiments were conducted successfully.

The rest of the paper is organized as follows. The second section describes WSTS’s basic terms and underlying concepts. The third section provides the description of two used algorithms (the backward reachability method and the Finite Reachability Tree). The forth section presents the architecture of the developed analysis tool. The fifth section shows how the developed tool is used for the analysis of Petri nets and provides performance analysis results. The sixth section summarizes and provides possible applications of the study for the future research.

II. Well-Structured Transition Systems

The definition of well-structured transition systems (WSTS) was proposed by Finkel in [2]. It is based on the two main concepts: transition systems (TS) and well-quasi-orderings on the states of these systems.

Transition system (TS) is one of the most general and widely-used models for formal description of the behavior of different systems. A *transition system* is defined by a structure

$TS = (S, \rightarrow)$ where $S = \{s, t, ...\}$ is a set of *states*, and $\rightarrow \subseteq S \times S$ is any set of *transactions* [3]. TS can be also supplemented by initial states, labels for transitions, durations or causal independence relations, and other information [3]; however, for the consideration of the concept of WSTS using of set of states along with labeled transactions is sufficient.

A binary relation $\leq$ on a set X is called *preorder* or *quasi-ordering (qo)* if it is reflexive and transitive. So for any $a, b, c \subseteq X$ we have:

1) $a \leq a$ (reflexivity);

2) $if\ a \leq b\ and\ b \leq c\ then\ a \leq c$ (transitivity).

Definition 1. A *well-quasi-ordering (wqo)* is a quasi ordering in which for every infinite sequence of elements $x_0, x_1, x_2, x_3, ... \subseteq X$ there exist such indices $i < j$ that $x_i \leq x_j$ [3, 6]. According to [7], there are several equivalent definitions of wqo; however, the definition given here is generally used in the WSTS theory.

Definition 2. A *well-structured transition system* (WSTS) is a transition system T$S = (S, \rightarrow, \leq)$ equipped with a qo $\leq \subseteq S \times S$ between states such that the two following conditions hold:

1) well-quasi-ordering: $\leq$ is a wqo, and

2) compatibility: $\leq$ is (upward) compatible with $\rightarrow$, i.e. for all $s_1 \leq t_1$ and transition $s_1 \rightarrow s_2$, there exists such a sequence of transitions $t_1 \rightarrow^* t_2$ that $s_2 \leq t_2$ [3].

$Succ(s)$ denotes the set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of s. Likewise, $Pred(s)$ denotes the set $\{s' \in S \mid s' \rightarrow s\}$ of immediate predecessors.

An upward-closed set is any set $I \subseteq X$ such that $y \geq x$ and $x \in I$ entail $y \in I$. A basis of an upward-closed I is a set I^b such that $I = \cup_{x \in I^b} \uparrow x$, where $\uparrow x =^{def} \{y \mid y \geq x\}$.

III. Algorithms

A. Backward Reachability Method

Backward reachability method proposed by Abulla et al. in [4] is intended to solve the covering problem which is to decide, given two states s and t, whether starting from s it is possible to reach a state $t' \geq t$. This is essentially one of set-saturation methods termination of which relies on the lemma that says that any increasing sequence of upward-closed sets ($I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$) eventually stabilizes (i.e. there is such a $k \in N$ that $I_k = I_{k+1} = I_{k+2} = \cdots$) [3].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$ and some upward-closed set of states $I \subseteq S$. Backward reachability method on the each j-th step generates the set of states from which I can be reached by a sequence at most j transitions [4].

More strict generalization was suggested by Finkel and Schnoebelen in [3], where it involves computing $Pred^*(I)$ as the limit of the sequence $I_0 \subseteq I_1 \subseteq \cdots$ where $I_0 =^{def} I$ and $I_{n+1} =^{def} I_n \cup Pred(I_n)$.

Definition 3. A WSTS has effective pred-basis if there exists an algorithm accepting any state $s \in S$ and returning $pb(s)$, a finite basis of $\uparrow Pred(\uparrow s)$.

The covering problem is decidable for WSTS if it has effective pred-basis and decidable $\leq$. The proof of this statement is given in [3]. Essentially, it is said that if there is a sequence K_0, K_1 ... with $K_0 =^{def} I^b$ (finite basis of I), $K_{n+1} =^{def} K_n \cup pb(K_n)$ and m is the first index such that $\uparrow K_m = \uparrow K_{m+1}$, then $\uparrow\cup K_i = Pred^*(I)$. By decidability of $\leq$, it is possible to check whether $s \in \uparrow Pred^*(\uparrow t)$.

B. Finite Reachability Tree

The Finite Reachability Tree belongs to tree-saturation methods which represent methods that consider all possible computations inside a finite tree-like structure [3]. It is also called the forward analysis method, in contrast to the backward analysis. Essentially, it is based on the ideas proposed by Karp and Miller in [5].

Assume there is some WSTS $TS = (S, \rightarrow, \leq)$. For any state $s \in S$, the Finite Reachability Tree is such a finite directed graph (tree) that:

1) nodes of the tree are labeled by states of S;

2) nodes are either dead or live;

3) the root node is a live node n_0, labeled by s (written $n_0 : s$);

4) dead nodes have no child nodes;

5) a live node $n : t$ has one child $n' : t'$ for each successor $t' \in Succ(t)$;

6) if along the path from the root $n_0 : s$ to some node $n' : t'$ there exists a node $n : t$ ($n \neq n'$) such that $t \leq t'$, we say that n subsumes n', and then n' is a dead node [3, 6].

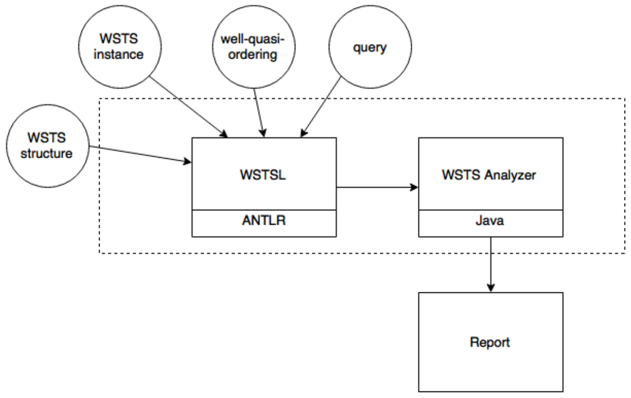
The Finite Reachability Tree is effectively computable if S has (1) a decidable $\leq$, and (2) $Succ$ mapping is computable [3]. All paths in the finite reachability tree are finite as any infinite path would include a covering node [6].

This algorithm can be applied to termination, inevitability, and boundedness problems (see [3] for details).

IV. Proposed Architecture

The general structure of the architecture of the developed tool is illustrated in Fig. 1. It consists of two main parts: Well-Structured Transition Systems Language (WSTSL) and WSTS Analyzer. Also there are four input parameters that are set by the user through WSTSL.

Fig. 1. Architecture of the developed tool



WSTSL is a programming language used in the developed system as the front-end which provides user with a means of describing the structure of WSTS and its essential operations and relations: $Succ$, $Pred$, preorder $\leq$. The supported data types are: integers, tuples, maps and sets. The analysis algorithms are implemented as the commands: *backwardanalysis()* and *forwardanalysis()*. As it is depicted in the Fig.1 the parser for WSTSL is built with the compiler-compiler Another Tool for Language Recognition (ANTLR) [8]. The WSTSL parser's sources are generated in Java.

WSTS Analyzer represents that part of the system which is responsible for the analysis of the transition system, which it gets from the WSTSL parser. WSTS Analyzer is implemented in Java, as it allows to naturally interact with the parser Java classes generated by ANTLR.

As it was noted above, the input that is provided by the user, consists of the four parts. Firstly, a general structure (WSTS structure) of the analyzed transition system should be described. Secondly, a well-quasi-ordering compatible with the defined structure should be specified. Then, a structure of a specific transition system (WSTS instance) that corresponds to the general structure is provided. Finally, the desired analysis algorithm with appropriate parameters (query) is invoked. Essentially, all these parts are described in a single input program written in WSTSL. Afterwards, the WSTS Analyzer runs the selected algorithm on the specified system and generates report which format depends on the choice of the algorithm.

V. Experiment

A. Petri Net

The applicability of the proposed approach could be demonstrated by an example of a Petri net which is well-structured transition system. The classical definition of this model is the following.

Definition 4. A Petri net (P/T-net) is a 4-tuple (P, T, F, W) where

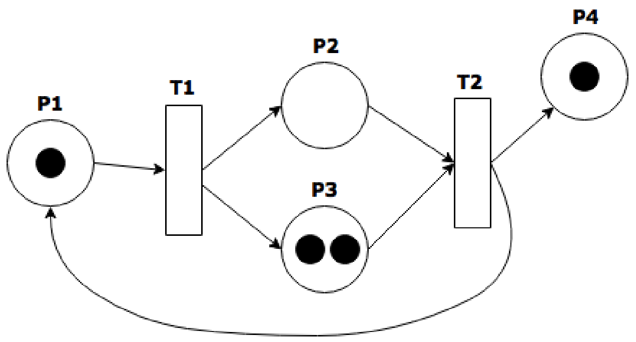
- P and T are disjoint finite sets of places and transitions, respectively;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs;
- $W : F \rightarrow \mathbb{N} \setminus \{0\}$ – an arc multiplicity function, that is, a function which assigns every arc a positive integer called an arc multiplicity or weight.

A marking of a Petri net (P, T, F, W) is a multiset over P, i.e. a mapping $M : P \rightarrow \mathbb{N}$. By $M(N)$ we denote the set of all markings of the P/T-net N.

We say that a transition t in the P/T-net $N = (P, T, F, W)$ is active in marking M if for every $p \in \{p \mid (p, t) \in F\}$: $M(p) \geq W(p, t)$. An active transition may fire, resulting in a marking M', such as for all $p \in P$: $M'(p) = M(p) - W(p, t)$ if $p \in \{p \mid (p, t) \in F\}$, $M'(p) = M(p) - W(p, t) + W(t, p)$ if $p \in \{p \mid (t, p) \in F)$ and $M'(p) = M(p)$ otherwise.

For the experiment, we will use the Petri net illustrated in Fig. 2.

Fig. 2. A simple unbounded Petri net



First of all, the general structure of the Petri net model described above should be defined by means of WSTL (Fig. 3).

Fig. 3. General structure of Petri net in WSTSL

```
type P                    : set of int;
type T                    : set of int;
type PT(P1:P, T1:T)       : set of [from P1,from T1];
type TP(P1:T, P1:P)       : set of [from T1,from P1];
type M(P1:P)              : map <from P1,int>;
type PN(P1:P, T1:T,
        PT1:PT, TP1:TP) : [P1,T1,PT1,TP1];
```

Secondly, we describe the specific Petri net instance in WSTSL (Fig. 4). PT1 and TP1 represent the arcs from places to transitions and vice versa, respectively. In tuples, defining arcs, the corresponding transition goes first for the convenience in the description of *Succ* and *Pred* function as it will be seen below.

Fig. 4. Description of the specific Petri net instance in WSTSL

```
var P1:P = {"P1","P2","P3","P4"};
var T1:T = {"T1","T2"};
var PT1:PT(P1,T1) = {["T1","P1"],["T2","P2"],
                     ["T2","P3"]};
var TP1:TP(T1,P1) = {["T1","P2"],["T1","P3"],
                     ["T2","P1"],["T2","P4"]};
```

Then, a well-quasi-ordering should be described (Fig. 5). As it is shown in [3], the element-wise ordering $M \subseteq M'$, when $M(p) \leq M'(p)$ for every place, is a wqo based on the Dickson's lemma [9]. Operator *forall iterator* | *test* generates a boolean value *true* if the condition *test* is met for each step in *iterator* and a boolean value *false* otherwise.

Fig. 5. Well-quasi-ordering definition in WSTSL

```
func wqo(PN1:PN, s1:M, s2:M)
    return forall p in PN[0] | s1[p] <= s2[p];
end func;
```

As it has been mentioned above in the Algorithms section, Backward Reachability Method requires effective algorithm for computation of *pred-basis*. The algorithm to compute it for Petri Net was suggested in [4]. How it is described in WSTSL is shown in Fig. 6.

Fig. 6. Description of the pred-basis and pred functions in WSTSL

```
func pred(PN1:PN, K:set of M)
    var P1:P = PN1[0];
    var T1:T = PN1[1];
    var PT1:PT(P1,T1) = PN1[2];
    var TP1:TP(T1,P1) = PN1[3];
    var predecessors: set of M(P1) = { };

    for s in K
        for t in T
            if forall tp in TP1[t] | s[tp[1]] - 1 >= 0 then
                s1 = s;
                for pt in PT1[t]
                    s1[pt[1]] = s1[pt[1]] + 1;
                end for;
                for tp in TP1[t]
                    s1[tp[1]] = s1[tp[1]] - 1;
                end for;
                predecessors = predecessors with s1;
            end if;
        end for;
    end for;
    return predecessors;
end func;

func pb(PN1:PN, K:set of M)
    return min(pred(PN1, I), wqo)
end func;
```

To state the covering problem, the initial state and the state which coverability is required to check should be specified. Afterwards, *backwardanalysis* function should be invoked with appropriate arguments (Fig. 7).

Fig. 7. Description of the initial marking and the marking which coverability it is required to check with Backward Reachability Method invocation

```
var m0:M(P1) = {<"P1",1>,<"P2",0>,
                <"P3",2>,<"P4",1>};
var mc:M(P1) = {<"P1",1>,<"P2",1>,
                <"P3",1>,<"P4",2>};

backwardanalysis(PN1,wqo,pb,m0,mc);
```

The tool provides the user with the output that contains sequence of sets K_i, where $K_0 = \{m_c\}$, $K_{n+1} = pb(K_n)$, their union $\cup_{i\in\mathbb{N}} K_i$ and its minimal elements (basis). Finally, it is reported whether the analyzed state (marking) m_c is covered or not (Fig. 8).

Fig. 8. Report of the tool for the backward analysis invocation

```
K0: [{P1=1,P2=1,P3=1,P4=2}]
K1: [{P1=0,P2=2,P3=2,P4=1},
     {P1=2,P2=0,P3=0,P4=2}]
K2: [{P1=1,P2=1,P3=1,P4=1}]
K3: [{P1=0,P2=2,P3=2,P4=0},
     {P1=2,P2=0,P3=0,P4=1}]
K4: [{P1=1,P2=1,P3=1,P4=0}]
K5: [{P1=2,P2=0,P3=0,P4=0}]
Union: [{P1=0,P2=2,P3=2,P4=0},
        {P1=0,P2=2,P3=2,P4=1},
        {P1=1,P2=1,P3=1,P4=0},
        {P1=1,P2=1,P3=1,P4=1},
        {P1=1,P2=1,P3=1,P4=2},
        {P1=2,P2=0,P3=0,P4=0},
        {P1=2,P2=0,P3=0,P4=1},
        {P1=2,P2=0,P3=0,P4=2}]
min(Union): [{P1=0,P2=2,P3=2,P4=0},
             {P1=1,P2=1,P3=1,P4=0},
             {P1=2,P2=0,P3=0,P4=0}]

The state {P1=1,P2=1,P3=1,P4=2} is not covered
```

As it has been mentioned above in the Algorithms section, Finite Reachability Tree requires effective algorithm for computation of *Succ*. How it is described in WSTSL is shown in Fig. 9.

Fig. 9. Description of the Succ function in WSTSL

```
func succ(PN1:PN, s:M)
    var P1:P = PN1[0];
    var T1:T = PN1[1];
    var PT1:PT(P1,T1) = PN1[2];
    var TP1:TP(T1,P1) = PN1[3];
    var successors : set of M(P1) = { };

    for t in T
        if forall pt in PT1[t] | s[pt[1]] - 1 >= 0 then
            s1 = s;
            for pt in PT1[t]
                s1[pt[1]] = s1[pt[1]] - 1;
            end for;
            for tp in TP1[t]
                s1[tp[1]] = s1[tp[1]] + 1;
            end for;
            successors = successors with s1;
        end if;
    end for;
    return successors;
end func;
```

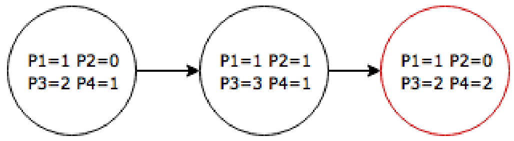
To construct Finite Reachability Tree only the initial state should be specified. Afterwards, *forwardanalysis* function should be invoked with appropriate arguments (Fig. 10).

Fig. 10. Description of the initial marking and the Finite Reachability Tree construction invocation in WSTSL

```
var m0:M(P1) = {<"P1",1>,<"P2",0>,
                <"P3",2>,<"P4",1>};

forwardanalysis(PN1,wqo,succ,m0);
```

The tool provides the user with the image which illustrates constructed Finite Reachability Tree (Fig. 11). Nodes are labeled with their states. Dead nodes are red. The node labeled with {P1=1, P2=0, P3=2, P4=2} state is dead since {P1=1, P2=0, P3=2, P4=2} ≥{P1=1, P2=0, P3=2, P4=1} (the latter state is represented by the root which subsumes the dead node labeled by the former state). It demonstrates that the place P4 is not bounded, as the red state has the P4 place large than P4 place in the initial state.

Fig. 11. Constructed finite reachability tree



B. *Lossy Channel System*

Another model that we considered was Lossy channel system (LCS) which is a subclass of FIFO-channel systems.

Definition 5. FIFO-channel system is a 6-tuple $(S, s_0, A, C, M, \delta)$ where

- S is a finite set of control states;
- $s_0 \in S$ is the initial control state;
- A is a finite set of actions;
- C is a finite set of channels;
- M is a finite set of messages (M^* is a set of finite strings composed of elements from M);
- δ is a finite set of transitions, each of which is represented by one of the following tuples $(s_1, c!\,m, s_2)$, $(s_1, c?\,m, s_2)$, (s_1, a, s_2), where $s_1, s_2 \in S$, $c \in C$, $m \in M$ and $a \in A$ (see below).

Transition $(s_1, c!\,m, s_2)$ changes the control state from s_1 to s_2, adding the message m to the end of the channel c. Operation $c!\,m$ is also known as a send action.

Transition $(s_1, c?\,m, s_2)$ changes the control state from s_1 to s_2, removing the message m from the beginning of the channel c. If the channel c is empty or its first element is not m, then this transition cannot occur. Operation $c?\,m$ is also known as a receive action.

Transition $(s_1, c?\,m, s_2)$ changes the control state from s_1 to s_2 and does not change the state of the channels.

In LCS it is also assumed that each message in some channels can be lost at any moment. To model this behavior one more operation $\tau(c, m)$ is introduced.

Transition $(s_1, \tau(c, m)\,, s_2)$ removes the message m from the channel c, and does not change the control state.

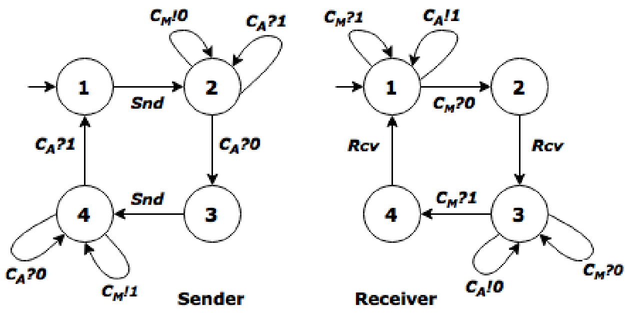
For LCS = $(S, s_0, A, C, M, \delta)$ the ordering $\leq$ is defined on the set of global states $\{(s, w) \mid s \in S, w{:}\, C \to M^*\}$ as follows:

$$(s, w) \leq (s', w') \Leftrightarrow s = s' \wedge w(c) \ll w'(c)\ \forall c \in C.$$

The ordering $\ll$ is a subword ordering: $u \ll v$ iff u can be obtained by erasing letters from v. It is shown in [6] that this ordering is a wqo; and, LCS systems are WSTS systems.

The concrete model that we considered was Alternating Bit Protocol (ABP). It is represented by Sender and Receiver which communicate via two FIFO-channels c_M and c_A. Sender sends messages to Receiver via c_M, while Receiver sends acknowledgements via c_A. Both channels can lose messages. Messages and acknowledgements contain one-bit sequence number 0 or 1. Sender continuously sends the same message with the same sequence number, until it receives an acknowledgement from Receiver with the same sequence number. Then, Sender changes (flips) the sequence number and proceeds with sending the next message. Receiver starts by waiting the message with the sequence number 0 (actually, it can initially send acknowledgments with the sequence number 1). When it receives such a message it starts sending acknowledgements with the same sequence number, until it receives the message with the flipped sequence number and so on. The described model is illustrated in terms of Lossy Channel System in Fig. 12.

Fig. 12. Alternating Bit Protocol modelled as a Lossy Channel System



The WSTSL definition of the structure and operators of LCS formalisms is analogously to the Petri Nets formalism WSTSL definition introduced in the Fig. 3-10.

C. *Performance*

To measure the performance of the implemented Finite Reachability Tree algorithm we applied it to the four different models, which include a model shown in Fig. 2 (Example 1) and the Petri Net models simulating the dining philosophers problem [10] for a number of philosophers equal to 5, 6 and 7. We executed the experiment on the following machine: Intel Core i7, 2.22 GHz, 16 GB RAM running OS X El Capitan (v. 10.11.6). *System.nanoTime()* method was invoked immediately before of the beginning of construction of a FRT and immediately after the end of construction, then the difference was calculated to measure run time for one run. In Table I in the Run time column average results for 20 runs are given in seconds. As well, sizes of the constructed FRTs are given. It can be seen that both run time and size of FRT grow exponentially for the philosophers problem instances.

TABLE I.

	Run time (s)	Size of FRT
Example 1	0.03596	3
Phil5	0.08587	241
Phil6	1.87815	25711
Phil7	5221.64756	88062003

VI. Summary

This paper address a lack of practical results in studies of well-structured transition systems. In order to fill this gap, there was presented one of the possible ways for development of the system capable to analyze WSTS with two common algorithms: backward reachability method and the Finite Reachability Tree. Well-Structured Transition Systems Language is introduced as a means of describing the user's input, which consists of the description of transition system's structure in general and specific instance's relations and values.

The tool can be used by researchers to investigate the efficiency of the implemented algorithms. It is expected that it is appropriate for conducting experiments on small and medium-sized WSTS. The technology eases the efforts required to check the potential of the WSTS analysis algorithms for practical applications and to make what-if experiments on newly developed formalisms.

The application of the tool is illustrated for the Petri nets and Lossy Channel System formalisms. Also, there were given results of the experiment on Petri nets modeling the dining philosophers problem. The performance analysis of the Finite Reachability Tree applied to this problem demonstrated the expected exponential growth of execution time; and, it indicates the need for further investigations of optimizations (e.g. reduction rules) that can be applied to make the algorithm effectively applicable in practice.

VII. Acknowledgements

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

References

[1] J. Burch, E. Clarke, K. McMillan, D. Dill and L. Hwang, "Symbolic model checking: 1020 States and beyond", *Information and Computation*, vol. 98, no. 2, pp. 142-170, 1992.

[2] A. Finkel, "Well structured transition systems," Univ. Paris-Sud, Orsay, France, Res. Rep. 365, Aug. 1987.

[3] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!", *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63-92, 2001.

[4] P. Abdulla, K. Čerāns, B. Jonsson and Y. Tsay, "Algorithmic Analysis of Programs with Well Quasi-ordered Domains", *Information and Computation*, vol. 160, no. 1-2, pp. 109-127, 2000.

[5] R. Karp and R. Miller, "Parallel program schemata", *Journal of Computer and System Sciences*, vol. 3, no. 2, pp. 147-195, 1969.

[6] E. Kouzmin and V. Sokolov, *Well-Structured Labeled Transition Systems*, Moscow: Fizmatlit, 2005.

[7] J. Kruskal, "The theory of well-quasi-ordering: A frequently discovered concept", *Journal of Combinatorial Theory, Series A*, vol. 13, no. 3, pp. 297-305, 1972.

[8] T. Parr, *The definitive ANTLR 4 reference*, Raleigh, NC and Dallas, TX: The Pragmatic Bookshelf, 2013.

[9] L. Dickson, "Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors", *American Journal of Mathematics*, vol. 35, no. 4, pp. 413-422, 1913.

[10] E. Dijkstra, "Hierarchical ordering of sequential processes", *Acta Informatica*, vol. 1, no. 2, pp. 115-138, 1971.

[11] Akshay S., Genest B., Hélouët L., Decidable Classes of Unbounded Petri Nets with Time and Urgency. In: Kordon F., Moldt D. (eds) Application and Theory of Petri Nets and Concurrency. PETRI NETS 2016. *Lecture Notes in Computer Science, vol 9698*. Springer, Cham

[12] Dworzanski L. W. Consistent Timed Semantics for Nested Petri Nets with Restricted Urgency, in: *Formal Modeling and Analysis of Timed Systems Vol. 9884*. Switzerland : Springer International Publishing, 2016. doi Ch. 1. pp. 3-18.

Stochastic Methods for Analysis of Complex Hardware-Software Systems

Aleksei Karnov
Institute for System Programming of
Russian Academy of Sciences
Moscow, Russia
Email: aakarnov@rambler.ru

Sergey Zelenov
Institute for System Programming of
Russian Academy of Sciences
Moscow, Russia
Email: zelenov@ispras.ru

***Abstract*—In this paper, we consider Markov analysis of complex software and hardware systems. Despite of its advantages, this method is barely used because of a large size of a model. Another problem is to translate system's architectural model to a Markov chain. So, we suggest a Markov analysis tool, including a translation algorithm, and some optional decisions to extremely accelerate the algorithm.**

I. Introduction

In this paper we consider a task related to verification of models of software and hardware systems. Such systems can be, for example, control systems for airplanes, ships, medical equipment, etc. The price of error in these systems is very high, but they are too complicated for "manually" analysis. Therefore such systems are modeled before implementation. On the stages of design, development, and verification of the models, it is necessary to constantly investigate system safety.

At present, three main methods of system safety assessment [1] are widely used: fault tree analysis, dependency diagram analysis, and Markov analysis. Each method has its own advantages and disadvantages. In this paper, Markov analysis is considered.

Markov analysis works with a Markov chain [2] - a stochastic process, which can be represented as a directed graph with weighted edges. Vertices of Markov chain represent different states, and edges are labeled by probabilities of a transition between states. The main drawback of Markov analysis is a size of Markov chains, which increases exponentially with number of components in the system. In addition, it is necessary to develop an algorithm, that takes system model and translate it to the Markov chain. These problems make Markov analysis not so popular as the other methods, and number of tools that use Markov analysis for complex systems is relatively small. However, such approach has its advantages: Markov analysis allows to look at the entire system, to consider not only causes and probabilities of certain single failure, but also ivestigate how various failures affect the system in the aggregate. Also Markov analysis, unlike the other approaches, allows to analyze self-recovering systems, since return to operational state is natural for Markov chains.

Thus, the task of development the Markov analysis tool of complex hardware-software systems is quite important and relevant.

II. Context

A. AADL and Error Model Annex

Architecture Analysis & Design Language (AADL) [3] is a language, that widely used for describing models of real-time hardware and software systems. Its features include description of both hardware (so-called execution platform) and software components of an analyzed system, and various connections between them. The models, described in AADL, may be used for documentation, for various kinds of analysis and for code generation.

Error Model Annex [4] is an extension of AADL, that allows to simulate appearance and propagation of errors in the system. For each component, a modeller can add a description of component's behavior states, for example, operational and failed. Transitions between system states are triggered by randomly occured error events and internal errors propagated from other components. An error propagation condition may depend on certain behavior state of the component, some error events, or error propagated from environment. Each propagated error has its own type, that allows to control what is exactly happened in the system. Also transitions between states can be defined implicitly - a state of some component may be a composite state of its subcomponents.

AADL and Error Model Annex together describe not only an architecture, but also error behavior of systems. It becomes possible to evaluate such properties of models as safety, reliability, the availability of its various states and ability to recover from them.

B. MASIW

MASIW [5] is an open-source framework for designing and analyzing of integrated modular avionics systems, that use AADL as a modelling language.

The project designed as plugins for Eclipse IDE, includes a variety of tools for working with AADL and Error Model Annex models. There is a big number of different analysis tools, for example, a fault tree analysis tool, but there is no Markov analysis tool.

C. Markov analysis

Any model subjected to Markov analysis must be represented as a Markov chain. A Markov chain can be represented in the form of a directed graph with vertices containing system states, and edges labeled with intensities of transitions between corresponding states. A Markov chain has the property of Markov process - a probability of a transition to any state depends only on a current state and a moment in time, and previous transitions are unimportant (can be characterized as memorylessness).

Markov models can be divided into models with discrete and continuous time, as well as time-homogeneous (also called stationary) and time-inhomogeneous. In time-homogeneous Markov chains, the intensities of transitions are constant, while in time-inhomogeneous Markov chains they depend on time. In time-homogeneous Markov chains, transitions occur according to the binomial (or fixed) distribution for discrete-time chains, and according to the Poisson distribution for continuous-time chains.

To determine the behavior of an analyzing system, it is necessary to specify a system of differential equations. The equations follow from the Markov chain. For all Markov processes (and a Markov chain, in particular) we have the Kolmogorov-Chapman equation [6]:

$$P^{(t+dt)}(S_j/S_i) = \sum_{k=1}^{n} P^{(dt)}(S_k/S_i)P^{(t)}(S_j/S_k) \tag{1}$$

This equation means that probability of a transition from state S_j to state S_i for some time $t + dt$ is equal to a sum of probabilities of passes into the target state S_i through all of intermediate states S_k.

Consider a time-homogeneous chain with an intensity of the transition between states S_i and S_j equal to $\lambda(S_i/S_j)$. Then for continuous-time Markov chains, the Kolmogorov-Chapman equation implies a system of differential equations

$$\frac{dP^{(t)}(S_j/S_i)}{dt} = \\ = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \tag{2}$$

And for discrete-time chains, a system of difference equations

$$\frac{P^{(t+\Delta t)}(S_j/S_i) - P^{(t)}(S_j/S_i)}{\Delta t} = \\ = -\sum_{k=1}^{n} \lambda(S_i/S_k)P^{(t)}(S_j/S_i) + \sum_{k=1}^{n} \lambda(S_k/S_i)P^{(t)}(S_j/S_k) \tag{3}$$

Denote by S_1 a certain initial state of the system, and consider equations (2)-(3) in case when $S_1 = S_j$. Denote by $P_i(t)$ the function $P^{(t)}(S_1/S_i)$. Then, the previous equations takes the following form:

$$dP_i(t)dt = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \tag{4}$$

$$\frac{P_i(t+\Delta t) - P_i(t)}{\Delta t} = \\ = -\sum_{k=1}^{n} \lambda(S_i/S_k)P_i(t) + \sum_{k=1}^{n} \lambda(S_k/S_i)P_k(t) \tag{5}$$

In addition, initial conditions appear

$$P_1(0) = 1, P_i(0) = 0, i = \overline{2, n} \tag{6}$$

Thus, we obtain the Cauchy problem [7]. The solution of this problem is a set of probabilistic functions of being a system in a definite state. This is the result of Markov analysis.

In this paper, we consider only the analysis of time-homogeneous Markov chains and models, as the most common ones. However, all results can be applied to time-inhomogeneous models, with the only difference being that intensities of Markov chain transitions depend on time, and they need to be stored as formulas, not as numbers.

III. Problem

The goal of this work is a development and implementation of a Markov analysis tool for complex hardware-software systems models within the MASIW framework. The tool takes input of some system model and a set of time points. At the output, the analyzer provides the probabilities of being the system in each of its possible states at moments of time, defined by user.

The main problem is to create a Markov chain on the basis of the original model. First, we need an algorithm that creates a correct Markov chain corresponding to the input data. Secondly, the result chain should be of acceptable size, so that the program can work for acceptable time in limited memory.

After a construction of a Markov chain, further action reduces to solving a Cauchy problem with a system of linear differential equations. An analytical solution of the Cauchy problem is too complicated, resource-intensive, and result is difficult to comprehend, so we use numerical methods.

IV. Solution

A. Markov chain

The primary task is to translate an AADL model into a Markov chain. In particular, it is necessary to find out what to regard as a chain node and what generates transitions between system states.

Obviously, the node must contain the state of the system, which is a combination of the states of all system components (the states of the components are described in the model as behavior states). However, if we take as a node any of all possible combinations, then the number of nodes will be no less than 2^n, where n is the number of system components. Real systems often contain more than 20 components, that, on the one hand, are few, but on the other hand, results in size of such Markov chain outside available memory.

We suggest the following solution of this problem. Let us exclude from the chain all unreachable states of the system, which, as practice shows, are the vast majority. First, some states of the system are unreachable by definition of an analyzing model. For example, the state of some component may completely depend on the states of its subcomponents. Accordingly, the component can not be in a failed state, while all its subcomponents are in operational states. Second, the failure of some components entails an almost immediate failure of others - for example, a breakdown of a processor entails a failure of all processes running on it. Thus, the state in which the processor is broken, but the processes on it are still working, though reachable in theory, at the very moment of the failure, but instantly replaced by another state.

Thus, we suggest the following approach. We assume that speed of error propagation between components is extremely small in comparison with time of system operation (which, in practice, is the case - for a unit of time measurement usually takes hour and even a day). Let us define a stable state of the sistem as a state, that does not change until new error events occur in the system and its components. We consider as nodes of designed Markov chain only the stable states of the system. The sets of arising events generate transitions between nodes of the chain.

For the sake of saving memory, we insert only reachable states to the Markov chain, and build it dynamically, from the initial state of the system, which is a combination of the initial states of the components. In each new node it is necessary to analyze transitions from the current state of the system. The state can change for some event or combination of events. So, we perform complete search for all possible sets of events - either of them can initiate a new transition. The probability of occurrence of each set of events is easily calculated, since each event contains information about its probability distribution. This is a multiplication product of probabilities of occurrence or negation of occurrence of each of the events, since all events are independent. The total probability of all sets of events, according to the law of total probability, should be equal to 1.

The algorithm is completed when all nodes of Markov chain are analyzed, starting from the node corresponding to the initial state of the system:

```
markovChain.addNode(initialStateNode)
queue.add(initialStateNode)
while !queue.isEmpty() do
  analyzeNode(queue.head())
  queue.add(newNodes)
end while
```

The analysis of each node of Markov chain looks like this: all possible sets of error events are searched, for each of them we calculate a stable state of the system into which the given set leads, and then a new transition (and, if necessary, a new node) is added to the chain.

```
for each errorEventSet in possibleSets do
  state = currentNode.getState()
  repeat
    watchedStates.add(state)
    state = calculateState(state, errorEventSet)
  until watchedStates.contains(state)
  node = markovChain.addNode(state)
  markovChain.addTransition(
  currentNode, node, errorEventSet.getProbability())
  watchedStates.clear()
end for
```

In the above algorithm, the state of the system is considered stable if we have already reached it before. This correctly handles the case when the state of the system has not changed - we have reached the same state as in the previous step. However, in theory, in a self-recovering systems, cycling may occur if an event with a failure and an event with component recovery occur simultaneously. With this condition, the loop stops, but this situation is not handled correctly. One of the main opportunities for further improvement of the algorithm is to improve the condition for achieving a stable state of the system.

B. *Calculation of new states*

In the previous paragraph, a general algorithm for constructing a chain was described, omitting the details of calculating new states of the system. To find out exactly how the system has changed, it is enough to go through all its components, and see what transitions between states are triggered for a given set of events and the current state of the system. The triggered transition is immediately applied to the system, and the algorithm step is completed.

```
for each componentState in systemState do
  for each compositeState in comp.getCompositeStates()
  do
    if checkStateExpression(compositeState.getExpression())
    then
      systemState.applyTransition(compositeState)
      return
    end if
  end for
  for each transition in comp.getTransitions() do
    if transition.getSource() == compState
    and   checkErrorCondition(transition.getCondition())
    then
      systemState.applyTransition(transition)
      return
    end if
  end for
end for
```

The checkStateExpression and checkErrorCondition functions check whether the transition condition is met. Such conditions can be interpreted as a logical formula, where variables corresponding to components behavior states, error events, and propagated errors, have value of true or false, depending on whether the system is in this state, whether an error event has occurred or whether an error of the specified type has propagated.

As soon as some component of the system changes its state, it means that we obtain a new state of the system, and the step of the algorithm is completed. If none of the transitions is triggered, then the system state has not changed, which is noticed by the algorithm described in the previous section.

C. Construction and solution of the Cauchy problem

After construction of a Markov chain, the final stage of the Markov analysis of the system is to construct a system of equations and solve the Cauchy problem. As mentioned earlier, each node of the Markov chain generates a differential equation (4) (or similar difference equation (5)). To save memory, it is not necessary to store the system of equations - the equation for any node can be easily constructed dynamically, passing through all transitions entering into this node and outgoing from it.

The resulting Cauchy problem can be solved by a numerical method from the Runge-Kutta [8] family of methods. In the analyzer, two methods are implemented: the Euler method, for discrete-time Markov chains, and the fourth-order Runge-Kutta method, for continuous-time Markov chains. The type of the chain is determined in advance, according to probability distributions of error events. An algorithm for calculating the variation of the function $P_i(t)$ on each time interval *delta_t*, taking into account the dynamic construction of the equation (Euler's method):

```
for each node in markovChain.getNodes() do
    i = indexOf(node)
    res = 0
    for each transition in node.getInTransitions() do
        k = indexOf(transition.getNode())
        res += transition.getProbability() * pPrev[k]
    end for
    for each transition in node.getOurTransitions() do
        res -= transition.getProbability() * pPrev[i]
    end for
    pCur[i] = pPrev[i] + delta_t * res
end for
```

Also, the value of the vector of probability functions $P(t)$ is saved at every time point defined by user. As soon as values at each necessary time point are calculated, the algorithm is completed.

D. Getting Analysis Results

Since number of system states in Markov chain can be very large, the result of analysis in the form of probabilities of being the system in each of them is practically impossible for reading. Considering that each system has its root component, we group all system states according to the states of the root component.

In this case, all the probability functions within the same group are summed up:

```
for each node in chainNodes do
    i = indexOf(node)
    state = node.getSystemState().get(rootComp)
    analysisResult.get(state) += p[i]
end for
```

After this, for each state of the root component, the probability of being the system in a this state at given time points is obtained. This is the desired result of the Markov analysis of the system.

E. Algorithm acceleration

Despite a partial solution of the problem of exponential growth of Markov chain size, the running time of full version of the algorithm still grows exponentially - due to a thorough search of all possible combinations of error events. Thus, we use some heuristics in the final program, accelerating the algorithm.

First, we limit the search of combinations of events. Since the probability of occurrence of one event is usually extremely small, the situation in which several events occur simultaneously is practically impossible. Therefore, a very small numerical parameter, limiting the probability of the combination of events under consideration, was added to the program. If the probability of occurrence of the set of events is less than this parameter, then the effect of the set of events on the system is not considered. This solution significantly reduced the running time of the program, without much loss of accuracy of the result.

The second solution relates to system's ability to self-recovery. In practice, there are few examples of self-recovering systems, and, in most cases, even a short-term failure of the system itself means fatal consequences. Accordingly, if the analyzed system has come to failed state, its further changes are not interesting to us - no matter what else can fail in the already failed system. Therefore, we introduce a set of states of the root component, that are considered as "absolutely" failed. If some node of Markov chain has failed state of the root component, then we do not analyze transitions from it. If analyzed system is not self-recovering, the result of the program remains the same, but is obtained in much shorter time.

Both modifications of the program are optional, as they may change final result in some cases, but their application reduces the operating time by several orders of magnitude. For example, a complete analysis of a system containing 24 components revealed 919 states of the Markov chain and took 1 hour. Limiting the frequency of the events considered by the number 10^{-30} gave a significant gain - the same set of states of the Markov chain and the same result of the analysis were obtained in 7 minutes. Since the system under test was not self-recovering system, the analysis with the stop-on-failed option was correct, and got the same result in 10 seconds. Setting relevant parameters allows to significantly accelerate work of the analyzer. One of the further options for improving the tool can be automatic detection and selection of optimizing parameters.

V. Related works

Markov analysis of AADL and Error Model Annex models is usually applied to systems consisting of only one component. Such algorithms doesnt consider error propagation mechanism and composite states, and limited by root component.

The tool from OSATE [9] framework, created for export AADL model into Markov chain model for PRISM [10] toolset, which provide further steps of Markov analysis, supports only the first nesting level of the component hierarchy and does not support different types of propagated errors. In addition, there were some problems associated with the syntactic correctness of the final PRISM model.

VI. Conclusion

In this paper we present a new Markov analysis tool, and in particular, an algorithm for translating AADL and Error Model Annex models into Markov chains. In addition, there were added some improvement for accelerating the algorithm, which make it possible to effectively use the tool in practice.

The presented tool can be further improved in various ways: adding support for time-inhomogeneous Markov chains, accelerating the work of the algorithm, changing some details of algorithm.

References

[1] "SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," Warrendale, USA, Dec. 1996.

[2] A. N. Shiryaev, *Probability (2Nd Ed.).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1995.

[3] P. H. Feiler and D. P. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*, 1st ed. Addison-Wesley Professional, 2012.

[4] P. Feiler, "SAE AADL error model annex: An overview," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep., 2014. [Online]. Available: https://wiki.sei.cmu.edu/aadl/images/1/13/ErrorModelOverview-Sept222011-phf.pdf

[5] "MASIW framework," https://forge.ispras.ru/projects/masiw-oss/.

[6] S. Kuznetsov, "Matematicheskie modeli protsessov i sistem tekhnicheskoi ekspluatatsii avioniki kak markovskie i polumarkovskie protsessy," *Nauchnyi Vestnik MGTU GA*, no. 213, pp. 28–33, 2015.

[7] J. Hadamard, *Lectures on Cauchy's Problem in Linear Partial Differential Equations (Dover Phoenix Editions).* Dover Publications, 2003.

[8] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM: Society for Industrial and Applied Mathematics, 1998.

[9] J. Delange, P. Feiler, D. Gluch, and J. Hudak, "AADL fault modeling and analysis within an ARP4761 safety assessment," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2014-TR-020, 2014. [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=311884

[10] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

Problems of Creation and Dynamic Analysis of Heterogeneous Models of Hardware/Software systems

Kamila Agaeva
Institute for System Programming of the Russian Academy of Sciences, High School of Economics
Moscow, Russia
kshagaeva@edu.hse.ru

Modeling is widely used for reasoning about different aspects of hardware/software systems. In this article the problem of analysis of such systems is considered, as well as studying of methods of working with tools of simulation of AADL models using MASIW toolset. A model of coffee machine was taken as an example of hardware/software system. Dynamic analysis using MASIW was performed on this model. Various kinds and methods of defining model components' specification were considered. The sequence of reasoning during formalization of model`s requirements was shown in the paper. One method of modeling of environment checks and specification of test scenarios was described. As a result, an architecture model of a coffee machine extended with behavior specifications on different levels of abstracts is presented.

Keywords — AADL models; dynamic analysis, behavior specification

I. Introduction

Currently modeling is an integral part of the design process of various systems. Thanks to modeling it is possible to consider a projected system with various aspects, each with only some fixed features and characteristics, that allows to abstract from minor details. In this article the process of modeling of software/hardware systems is considered. Such systems are a bunch of several components that process and interpret data.

A. Analysis of hardware/software systems

A constructed model of hardware/software systems can be used to further analysis of the entire system`s characteristics or behavior, based on the information from the operation of components. This makes it possible to draw conclusions on system effects. Such systems can be researched while using static or dynamic analysis, as well as by analysing the structure and behavior of the system`s functioning.

B. Types of model analysis

Static analysis considers a mathematical description of the model`s components, which is compared to the formulated requirements. Due to this comparison, it is possible to calculate the characteristics of the components (for example, an assessment of the mode of interaction with other components or the time of the system's response to external events). Such attitude can help to draw conclusions about the correctness or incorrectness of the analysed model in the future.

Carrying out the dynamic analysis requires making sure that there are certain regularities involving the model changes. For this purpose, the model components must have an executable behavioral specification, and work correctly with time model. In this case, the analysis provides data on new states of components and new relationships of their interactions.

The analysis of behavior is based on the consideration of characteristics that are manifested during the components` interaction and the environment. These data are also associated to the interaction of the component with the environment or with its response to emerging events. Structural analysis, in its turn, works with those characteristics that are associated with the location of components, their connection and the method of communication.

1) Specification of model components

The characteristics of the model components necessary for the analysis are specified by their specification, which can be conditionally divided depending on the properties they express [1]. Thus, the properties can be divided into several types:

- functional properties describing changes in states and considering the relationship between input and output data;
- temporal properties express indicators that can be determined over several states of the system or can provide a relationship between system events and state transitions;
- resource properties, whereby there are restrictions on such resources as time or space that can be used by the operation or at the time of occurrence of events.

Such separation allows us to identify the specifications expressing the functional, temporal and resource properties respectively. Another classification is based on the separation of specifications, depending on the described artifacts [1]. These can be specifications related to the requirements, interface, analysis or architecture of the system.

C. Problems of developing of multicomponent systems

The recorded specification for each hardware/software systems` component is a large amount of information.

However, human capabilities are not enough for analysis. One way to solve this problem is to use automated support tools. For example, it is possible to formalise information about the system, by translating it into a unified machine-readable form, which makes it possible to automate its processing.

D. *The AADL language*

At the moment, various design and analysis languages, such as Marte [2], SysML [3], and architectural analysis and design language (AADL) [4] are used to represent information about the projected system in a machine-readable form. In this paper we will focus on the usage of (AADL). It is used for modeling hardware/software architecture of embedded real-time systems. AADL allows to analyse computer systems on such aspects as fault tolerance, security, security, performance, etc. [5]. The architecture of the system is considered as a hierarchy of interacting software and hardware components. In this case, each component of the system can be expanded by adding the user properties. In addition to the extensibility of this language, the advantage of AADL is ability to work with different levels of abstractions. Thus, the architecture of the system can be built step by step. It can start with the most general view, gradually lowering the level of abstraction, then refine the internal organisation of components and introduce additional properties. In this case, we get a correct and full-fledged model from the point of view of AADL and from the point of view of defining the architecture. This can help to design and analyse the heterogeneous models that represent a complex set of different components that can appear at different levels of abstraction.

1) *Instruments supporting the AADL language*

The creation and analysis of models written in AADL can be done using several tools. The most widely used examples are presented below.

OSATE 2 is an open source platform supporting the AADL language. It presents AADL text and graphics editors, as well as sets of some analysis tools (for example, BLESS [6], Error Annex V2 [7]). In addition, open source code allows developers to add their own plug-ins.

Ocarina is toolset, written using Ada for managing AADL models. Its capabilities can be described as:

- syntax analysis of AADL models;
- semantic analysis;
- code generation using Ada and C;
- checking for model compliance by comparing models written in AADL with Petri nets;
- calculation of execution time in the worst case using Bound-T [10];
- using REAL [11] to check the properties and metrics of AADL models.

AADL-inspector is the environment for processing of AADL models. Its goal is to provide an easy-to-use and extensible tool for performing static and dynamic analysis of AADL architectures and a simplified way to connect any AADL compliance tool or code generator. Its main capabilities include:

- an AADL syntactic analyzer;
- a set of static semantic rules checkers;
- schedulability analysis (using Cheddar [13]);
- virtual execution of AADL models based on Behavior Annex (Marzhin [14]).

MASIW is open source integrated development environment for design and analysis of AADL models. This toolkit was developed by ISP RAS jointly with GosNIIAS within the state development program of Integrated Modular Avionics. The main features of this tool are presented below:

- creating and editing models in AADL using text and graphic editors;
- model analysis
  - analysis of the structure of the hardware/software complex;
  - checking the constraints using REAL, PyCL [15];
  - analysis of data transmission characteristics in the AFDX network [16]; (2011 - 6)
  - simulation of the model with supporting for heterogeneous models [17];
  - analysis of propagation of error and probabilities with FTA and FMES;
- generating model data
  - generation of processor scheduler;
  - generation of configuration data;
- ability to reuse AADL-libraries of third-party developers;
- use of the concept of representations, which allows to adapt the created model of the system to a form abstraction from the way of implementing this model, but containing the most relevant data.

In the context of this work MASIW was chosen as the tool working with the AADL language.

II. Using the MASIW simulator

Among available tools of MASIW simulator [18]. It allows to analyse the behavior of the system and its components was considered.

A. *Behavior specification in the MASIW simulator*

The simulator provides support for standard ways of specifying the behavior specification for model components that are located below:

1) *Behavior Annex*

This approach is a standardised extension of AADL, which allows to describe the behavior of components in terms of a nondeterministic finite state machine. Each component has several states and a set of actions. Also, there are transitions between each state that have some condition. Such condition helps to check on the performance of any inequality or expecting an external event (for example, wait for an event arriving at the input port of this component). Moreover, the

Behavior Annex allows you to define a response to an external event and data sending.

2) Java classess

An executable model can be specified using Java code. For this purpose, the MASIW simulator has Java-library, that provides additional tools for creating of behavior specification of model. Java-code allows to work with model time, as well as to react to the external events. For example, you can consider a component with a behavior specification and a system of the surrounding world in which the component is placed. In the absence of an external system, the component has a certain algorithm of its behavior. But with the emergence of an environment that can affect it, his behavior can change. Java-modeled behavior specification also allows to handle such situations.

3) ARINC-653

ARINC-653 [19] is the standard that represents a set of services for the safe design of avionics application software.*B.* This standard presupposes that the software system as a divided into several parts the composition of applications or partitions. Each partition is isolated in terms of time and space and works as if it was running on the same processor. This technique is necessary to process and isolate errors that occur in some partitions. ARINC-653 partitions can be used in the MASIW simulator as one of the options to specify the behavior of components.

B. Tracing data in the MASIW simulator

The purpose of analysing the behavior of the model is to keep track of the correct functioning. For convenient observation, the MASIW simulator automatically traces the data about the transmitted messages within the system. Tracing data also provides the opportunity to use special systems, or oracles, that help determine the expected results for comparison with actual results. This feature provides information about the time of sending such messages, data sources and, possibly, the history of the messages. In addition, this approach allows developers to monitor the internal state of the components.

For convenience of work with the received trace data, the simulator of MASIW allows to aggregate the received information into reports. Such reports allow to select from the entire set of events only those that relate to checking of the conditions for meeting the requirements. This report is also used by the MASIW simulator to demonstrate:

- which validity conditions have been tested;
- which components have been checked for correctness conditions;
- what is the status of the component check (which components are found to violate the correctness conditions and which ones are not);
- the cause of the violations that occurred.;

III. The target and objectives of work

Within this work, the goal was to study the work with the AADL modeling tools and analysis tools of the MASIW simulator on the example of a model of a coffee machine. To achieve this goal, it is necessary to perform several tasks:

- the formalization of requirements and test scenarios for the coffee machine being modeled;
- construction of the architecture of a model of the coffee machine, based on the formalized requirements using the MASIW simulator;
- adding the behavioral component to the elements of this model;
- use of model behavior specifications at different levels of abstraction (Behavior Annex, java classes, ARINC-653);
- analysis of the conducted experiments;

IV. Description of sequence of work

A. Formalization of requirements

Even the simplest coffee machine includes many components, such as pistons, pressure sensors or pumps. In this work, a simplified version of the model of the coffee machine was chosen at level of abstraction, which would allow one to abstract from the complex architecture of the coffee machine and include the main hardware components, the implementation of which would allow accomplishing the set of tasks. The same goes for the software part. There are many algorithms in the real machine, but in the created model, for simplicity, the processing of the order from the user and notification of the user about the readiness of the drink are considered. In addition, the created model contains the basic requirement that determines its purpose, which is the preparation of a drink: coffee or cocoa. Besides, additional requirements have been formulated, which relate to a more precise level of abstraction. (нужно подвести к требованиям)

a) The coffee machine must receive orders from the user, issue the corresponding drink as the result and inform the user about it.
b) The coffee machine must have a button for making orders to the coffee machine.
c) The coffee machine must be in three modes: "cooking coffee", "cooking cocoa" and "shutdown mode."
d) The user has the ability to only order coffee or cocoa.
e) The user has the ability to change the mode of the coffee machine with the control knob. It allows the user to select a drink that the coffee machine must cook, or turn coffee machine off.
f) The coffee machine cannot change its own mode
g) The coffee machine must not prepare drinks without user order.
h) If user want to make order, it is necessary to press the button and the coffee machine must be in the "coffee preparation" or "cocoa cooking" mode.
i) If user made an order, the result must be received 31 seconds made.

j) The coffee machine must be ready to accept the next order only after the processing of the previous order is completed.

k) Order processing must mean either the coffee machine must issue an order or ignore it if the coffee machine is in the "shutdown mode".

l) If user makes several orders faster than once in 31 seconds, the coffee machine will process each order made.

m) If user makes several orders within 31 seconds, the coffee machine must execute only first order, ignoring all the others.

B. Description of test situations

Test situations have been developed to consider tested system in different situations, checking requirements. Test situations, which are presented below:

a) Testing user's nominal behavior, in which the user orders one drink (coffee or cocoa). As a result, it is expected that the coffee machine will give a signal of completion and drink itself.

b) Testing the case in which the user makes several orders during 31 second. It is expected to receive a response from the system of the coffee machine about the readiness of only one drink for 31 seconds, as well as getting the finished result.

c) Simulation of the situation in which the user makes several orders of coffee or cocoa once in 31 (or more) seconds. In this case, the number of drinks and signals of completion should be the same as the number of orders made by the user. At this time from the acceptance of each of the orders to the receipt of the result should not exceed 31 seconds.

d) Testing the situation in which the user does not perform any actions. In this case, the coffee machine should not notify and give out anything.

C. Basic concepts of AADL

The following basic concepts were used to design the interface and system architecture using the AADL language.

In AADL, all the components you create can be divided into several categories. Some of them, that have been used in this work are bellow.

A system is a common container of software and hardware components, as well as other related systems, with a fully or partially known internal structure.

A device represented as an active hardware node with a certain functionality, the internal structure of which is not disclosed.

Data component that represents the abstraction of a variable type in the program code. Some components (devices and systems, for example) may require access to such components. For this purpose, additional interface features that explicitly require access are added to them. It is possible to set the data access rights.

Processes in AADL denote a category of components that corresponds to a separate virtual address space and can be a complete program unit if it contains at least one control flow.

Thread are also a type of AADL component. They are an executable instruction sequence, which, from the processor's point of view, is provided as a single dispatch object.

The processor is represented in the form of the main calculator, which deals with scheduling and executing of program code.

In addition, ports provide support for communication between components. They can be divided into several types: event data port, event port, data port. The first type of ports allows you to send or receive control and data, the second one allows you to send only control, and the third allows you to transfer only the data.

D. Designing the architecture of the system environment

The creation of the architecture of the coffee-machine model began with the determination of the level of abstraction of given system. This helps to identify which components will be used to design and further analyse the interface of the model of the coffee machine. We can distinguish some details from requirements:

- availability of a way of interaction with the user through the button;
- the coffee machine can prepare coffee and cocoa;
- the coffee machine informs you that the drink is ready.

These details were considered in building the environment of the coffee machine by using the following components (Figure 1) in the external system "environment_ext".

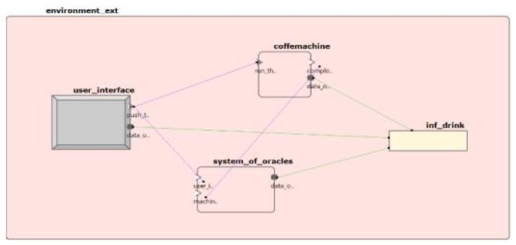


Figure 1 – external system envirnonment_ext

The system of the coffee machine, which at the initial stage of building the interface was seen as a black box, capable only of interacting with the outside world. It has an input event data port, "run_the_process", output event port "completion", which were modeled according to the requirements (Figure 2).

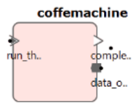


Figure 2 – system of coffee machine

The device "user_interface", which is responsible for user behavior; with the help of the output event data port "push_the_button" it simulates pressing the button that sends the order (the name of the desired drink is sent as the message: coffee or cocoa).

The system "system_of_oracles" is one of way to formalize requirements. It allows to verify the properties. It receives information about the user’s choice through the event port "user_inf" and information about the availability of the order through the input event port "machine_inf". Such a connection allows to track temporary information about the service of the order;

Data component "inf_drink" (моделирует)simulating in the given system a certain physical state, which gives information on the amount of ingredients used in the process of preparation of the drink. It is possible to change this state by the system of the coffee machine.

It is worth notice that both the user and the system of the coffee machine are modeling of real-life notions. If we talk about the oracle system, this component is artificially added only to ensure the possibility of checking certain properties. The same applies to the formation of a physical state using the “inf_drink” component.

Designing the interface of the coffee machine with the help of the components listed above allows to proceed to build the architecture of the external world of the coffee machine system.

E. *Designing the architecture of the model of oracles*

The modeled oracle system is a set of oracles, each of which tests a given requirement (Figure 3). Use oracles allows to formalise the requirements mentioned above. The oracles are presented in the form of a device. Some of them receive input from the user via the event port “user_inf”, data about the completion of the coffee machine through the event port “machine_inf” and some have access to reading data from the inf_drink component.

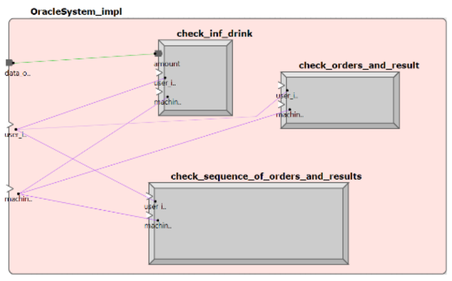


Figure 3 – system of oracles

Using this type of ports allows to track the time intervals during which the order was processed. Thus, with the help of these oracle devices, the following properties were tested:

- if two orders are received, the coffee machine returns two results;
- at least one order must be received before the result is received;
- the situation in which the user will not be able to prepare the coffee machine himself.

Thus, oracles allow to cover the verification of all the requirements presented in the above.

F. *Design the architecture of the model of user*

The device simulating the user's behavior includes several of its implementations, providing verification of the test situations. The options are below:

- order two cocoa beverages with a time difference of pressing the button in 2 seconds;
- order two cocoa drinks with a time difference of pressing the button at 31 seconds;
- order three drinks ("cocoa", "coffee", "cocoa") with a time difference of pressing the button at 1 and 2 seconds respectively;
- order two "coffee" drinks with a time difference of pressing the button for more than 31 seconds.

To verify compliance with requirements and perform simulations using the MASIW simulator, the components of the oracle and user system were given behavioral specifications. It is worth notice that for this purpose the Behavior Annex was chosen, which is due only to the convenience of representing such behavior with the help of these means. (возможно было использовать и другие, но используем это)

G. *Behavior specification for components of the environment of the coffee-machine*

By using the Behavior Annex capabilities, behavioral specifications have been modeled for the oracle system devices and for the user.

User behavior was described by simulating the sending of a different number of orders through the output port, as well as the use of temporary delays between them to check requirements that related to the need for time intervals (requirements i, l and m).

Behavior specifications for oracles use the data trace function. This provides a convenient way to get a response about the feasibility of a given property by referring only to the generated simulation report.

The modeled architecture and the behavior of the coffee machine’s environment allow to proceed to a more complete specification of the coffee machine’s system itself.

H. *Building the architecture of the coffee machine system*

The construction of the architecture of the model of the coffee machine consisted in the addition of software components (processes, flows, data) and components responsible for the physical resources of the system (processors, devices). The created architecture must be in accord with the formalized requirements and the installed interface. Therefore, there was considered the case in which the model of the coffee machine was limited to the presence of only three devices: milk dispensers and coffee, a water heater and a valve. These elements represent the hardware of the coffee machine. As a program part, processes and a processor are presented that control the distribution of input and output data.

Moreover, processor that located in system of coffee machine can manage processes. Also, there are devices whose action on the outside world is modeled through data access.

The next stage in modeling the system of the coffee machine was to add the behavioral component to some of its components.

I. *Using the Behavior Annex to specify the behavior of the components of the coffee-machine*

Due to the capabilities of the Behavior Annex, behavioral specifications have been modeled for all devices entering the system of the coffee machine, as well as for threads.

The coffee and milk dispensers, as well as the water heater and its valve, have almost identical behavioral specifications. They are triggered when an input message or event is received, then they work out a certain amount of time (each device works for a different period), and then send an event or message through its output port.

If we talk about control flows, then the process of the received message about the type of drink was included in the specification of the behavior of the control flow "data_processing". The result of this process determined the data to be sent to the two dispensers of the coffee machine.

At the same time, the specification of the behavior of the control flow "waiting_result" was based on the expectation of receiving events about working out all devices for the model of the coffee machine`s system, after which a signal was sent as a result about the completion through the output port.

Simulations carried out with the help of specified behavioral specifications for oracles, users and components of the coffee machine made it possible to verify those requirements that were formulated at the initial design stage, as well as to simulate all test situations and obtain the result in the form of reports generated by tracing. At the next stage of the work, another level of abstraction was considered, in which the behavior specification was also specified by the processor “cpu”. This was implemented using the Java language.

J. *Using the Java language as a way to specify a behavior specification*

The cpu processor created in the model is associated with the operation of two processes: “data_processing” and “waiting_result”. This processor should monitor the arrival of the message in one of the process ports and give control to the desired one. Behavior Annex doesn’t have tools that can simulate such behavior. That’s why the Java language is used.

To simulate this behavior, a special library was developed in the MASIW simulator. The basic ideas that make it possible to generate a behavior specification for a component are presented below. The SimulationManager interface can be referred to the basics of this library. It provides access to the formation of behavior that can be actively used in modeling. Thus, each component has the ability to:

- sending messages between components using:
  - asynchronous (non-blocking) messages; this method does not return anything and provides instant control return;
  - synchronous (blocking) calls; in this case, the call itself can cause the execution of additional procedures and only after returning some result and control;
- granular approach to processing incoming messages and blocked calls; this method places the processing process on several handlers, some of which can be added by the developers themselves;
- setting the waiting time for the calling thread; This feature is like using the computation () function in the Behavior Annex;

If the component under consideration has managed components, then there are several more possibilities for it:

- determine the desire or unwillingness to manage a component;
- receive alerts that there will be something happening with the managed components.

In situation with coffee-machine, an event handler was added for this processor. It helped track the port in which the event came. After that, thanks to the list of the binding elements connected with the processor, it is possible to find the necessary component and give it the right to work.

K. *Using ARINC-653-partitions to specify a behavior specification*

When forming a component behavior specification, we can consider the case in which the highest level of abstraction is considered. At this level, the program, represented as ARINC-653 partitions, itself becomes its own specification of behavior. This approach could be used as one of the options for compliance testing.

V. Conslusion

The work carried out on the construction and analysis of the model of the coffee machine with the help of simulation tools of the MASIW allowed to draw several conclusions:

- The process of forming of requirements for the developt model depends on the level of abstraction on which this model is considered. This affects what components will be included in the system, their functionality and possible behavior.
- For completeness of checking, the simulated external environment should provide an opportunity to verify not only nominal behavior that meets all the points of the requirements, but possible inadequate options, in order to confirm the correctness of processing such situations.
- Use and support of heterogeneous models allows to consider systems in which components are at different levels of abstraction: some of them are already fully described, and some are at the initial stage of development

References

[1] John Hatcliff, Gary T. Leavens, K. Rustan M. Leino, Peter Muller, Matthew Parkinson. Behavioral Interface Specification Languages. ACM Computing Surveys. vol. 44, no. 3, article 16, June 2012

[2] Object Management Group (OMG). “UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems.” Version 1.1.

[3] Object Management Group (OMG). “Systems Modeling Language SysML”, Version 1.3.

[4] SAE International. “Architecture Analysis & Design Language (AADL)”, SAE International Standards document AS5506B, Nov 2004, Revised Mar 2012.

[5] A.E. Platunov, N.P. Postnikov. Vysokourovnevoe proektirovanie vstraivaemykh system. – Sant-Petersburg, IFMO, 2011, vol p.41-42 (in Russian).

[6] Brian R. Larson, Patrice Chalin, and John Hatclif. BLESS: Formal Specification and Verification of Behaviors for Embedded Systems with Software», Kansas State University, USA.

[7] SAE International Standards Document A55506/1A Arhitecture Analysis and Design Language (AADL): Error Model Annex, 2015.

[8] Jerome Hugues[electronic resource]. -PolyORB-HI/Ada runtime, Access mode: https://github.com/OpenAADL/polyorb-hi-ada, free.

[9] Jerome Hugues[electronic resource]. -PolyORB-HI/C runtime, Access mode: https://github.com/OpenAADL/polyorb-hi-c, free.

[10] Niklas Holsti, Sami Saarinen. Status of the Bound-T WCET Tool». Space System Finland Ltd.

[11] O. Gilles, J. Hugues. Expressing and Enforcing User-Defined Constraints of AADL Models, Engineering of Complex Computer Systems (ICECCS), 2010.

[12] P. Dissaux, F. Singhoff. Stood and Cheddar: AADL as a Pivot Language for Analysing Performances of Real Time Architectures, Proceedings of 4th International Congress ERTS-2008.

[13] Christophe Ponsard, Philippe Massonet, Gautier Dallons. From Rigorous Requirements Engineering to Formal System Design of Safety-Critical Systems. Safety-Critical Software, 2008.

[14] Pierre Dissaux, Olivier Marc. Executable AADL: Real-Time Simulation of AADL Models, 2014.

[15] Alexey Khoroshilov, Eugene Kornykhin. PyCL – Python-based AADL Constraint Language, - Madrid, Spain, 2015.

[16] O. Khorevsky, A. Khoroshilov, A. Ugnenko, S. Zelenov. Configurable AFDX network simulation. In Proceedings of the International Space System Engineering Conference DASIA-2011, pp. 31-34, San Anton, Malta, May 17-20, 2011

[17] Denis Buzdalov. Simulation of AADL models with software-in-the-loop execution, 2016.

[18] Buzdalov D.V., Zelenov S.V., Kornykhin E.V., Petrenko A.K., Strakh A.V., Ugnenko A.A., Khoroshilov A.V Instrumental tools for the design of integrated modular avionics, 2014.

[19] ARINC Specification 653P0-1. Part 0: Overview of ARINC 653, 2015.

Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions *

Anton Volkov
Lomonosov Moscow State University, GSP-1 Leninskie Gory, Moscow, 119991, Russian Federation
Mikhail Mandrykin
Institute for System Programming, 25 Alexander Solzhenitsyn st., Moscow, 109004, Russian Federation

***Abstract*—Software verification is a type of activity focused on software quality control and detection of errors in software. Static verification is verification without the execution of the software source code. Special software – tools for static verification – often work with program's source code. One of the tools that can be used for static verification is a tool called CPAchecker. The problem of the current memory model used by the tool is that if a function returning a pointer to program's memory lacks a body, arbitrary assumptions can be made about this function return value in the process of verification. Although possible, the assumptions are often also practically very improbable. Their usage may lead to a false alarm. In this paper we give an overview of the approach capable of resolving this issue and its formal specification in terms of path formulas based on the uninterpreted functions used by the tool for memory modeling. We also present results of benchmarking the corresponding implementation against existing memory model.**

***Index Terms*—memory model, predicate abstractions, static verification**

I. Introduction

Software verification is a type of activity focused on software quality control and detection of errors in software [1]. Static verification is a verification without the execution of the software source code.

Special software – tools for static verification – often work with program's source code. Depending on the tools used for static verification it is possible to conduct analysis of the said source code to search for errors in program's behavior.

One of the tools that can be used for static verification is a tool called CPAchecker. It takes program's source code as an input, creates a CFA (control-flow automaton) and uses it to run the analysis. One of the analyses the instrument is capable of is a reachability analysis. In this paper we consider reachability properties that can be expressed as checking if the call to an error function is reachable. Its strong side is that the CPA (configurable program analysis) [2] concept allows to use several analyses at the same time for program verification. The tandem of Value Analysis and Predicate Analysis produces good results in terms of verification precision / verification time ratio.

II. Definitions and notations

We will call *a model of program's memory* a strategy of organization and representation of program's memory. By *region* we will refer to a set of lvalues with the following restriction: if two lvalues are taken from two different regions they necessarily reference disjoint memory locations [3]. For example, different regions may be safely assigned to the lvalues referring distinct structure fields under the following conditions:

- no address taking of the fields is present in the program's source code;
- the fields do not become targets of some pointers by the usage of pointer type conversion or address arithmetic.

The situation when a program's error state is reachable because of the errors in verification tools is called *false alarm.*

III. Task

Existing memory model employed by Predicate Analysis of the CPAchecker tool uses uninterpreted functions. Each of those functions has only a name and a number of arguments. If $f(x)$ is an uninterpreted function, a and b are any of its arguments for which $a = b$ is true then $f(a) = f(b)$ [4]. Uninterpreted functions in the CPAchecker tool are used to establish a correspondence between a memory location and the value stored at this memory location. Depending on the type of the expression different uninterpreted functions should be used.

Existing memory model of the CPAchecker tool uses typed regions. This means that all lvalues of the same type exist in the same region. However a large number of lvalues of the same type is present in any big enough program written in the C programming language. This leads to the addition of a big number of logical constraints for each event of a pointer's memory update. The constraints express checks for potential equality of the updated lvalue to each memory location in the region. Those checks allow to determine precisely what memory should also be updated but noticeably increase the length of path formulas.

The problem of the current memory model used by the tool is that if a function returning a pointer to program's memory lacks a body, arbitrary assumptions can be made about this function return value in the process of verification. In other words it is considered possible for this pointer to point at any lvalue in the region. Although possible, it is also practically very improbable. In those cases it is hard to determine if a path leading to an error label really does or doesn't exist. One of the approaches capable of resolving this issue suggests

* The research was supported by RFBR grant 15-01-03934

the introduction of smaller regions that divide a bigger typed region.

IV. B&B MEMORY MODEL

A. Memory model overview

B&B memory model was proposed by Richard Bornat and had been based on the work of Rod Burstall [5], [6]. It is used in Frama-C verification tool in Jessie plugin which is capable of performing verification of the C programs. In its foundation are assumptions that can introduce regions of smaller sizes instead of having very big one for a type. These assumptions state that if struct data type fields never occur as arguments to the address taking operator (`&`) in program's source code then those fields can be placed to separate regions. Otherwise they must belong to the same region as the normal pointers of the same type.

This memory model has some flaws. It does not take into account that the struct fields can be accessed through address arithmetic and pointer conversions. It also needs mentioning that some overhead costs are required for region support.

Taking into account the pros and cons of the model it is possible to say that the B&B memory model looks promising.

B. Formal specification

For ease of specification we will assume the following:

- variables can only be of struct s * types;
- struct s fields can only be of int type;
- struct s has n fields: struct s { int f_1, f_2, ..., f_n; };

Program's memory location can be represented by an lvalue expression like pointer dereference. To model changes to the program's state when assignments to lvalues arise the CPAchecker tool uses uninterpreted functions [4].

We assume absence of pointer arithmetic and restrict pointer dereferences to the applications of the arrow operator ($p \to f_i$, where p is a pointer to the struct type and f_i is one of the struct fields).

Let Υ be a set of uninterpreted functions. Its elements are uninterpreted function G that is used for accessing a memory location in global region, a finite number of uninterpreted functions F_i, where each function F_i represents the state of the memory region corresponding to lvalues of the form $b \to f_i, i = \overline{1,n}$ and the uninterpreted function $undef\_ptr$ with zero arity that models the usage of the program's functions returning an unknown pointer.

Let $B(e)$ be an uninterpreted function used for global memory location modeling and $B^i(e), i = \overline{1,n}$ —a finite set of uninterpreted functions used for memory location modeling in regions corresponding to F^i uninterpreted functions. For address representation it is suggested to use expressions like a, where a is a variable. The axioms of the memory model (positivity of addresses and their non-intersection within one region) can be represented as follows:

- $a > 0$;
- $B(a) = k$, where k is a unique number for each such variable.

The tool uses SSA representation to model the varying state of program variables and memory regions. In this representation usage of a name splits into usages of its versions. Each time an assignment happens to a program variable or a memory region represented by the corresponding variable or uninterpreted function in the path formula, the version number (index) of that variable or uninterpreted function increases.

Let $Index : \Upsilon \to \mathbb{N}$ be a mapping of a set of uninterpreted functions Υ to a numerical set of their indices.

Let $Alloc : \Upsilon \to Addrs$ be a mapping of a set of uninterpreted functions Υ to the set of subsets of memory locations $Addr$: $Addrs = 2^{Addr}$.

We will use a supplementary function mem_upd :

$$mem\_upd(p, f, m', m) = \bigwedge_{a \in Alloc(f)} \big((p = a) \vee (f_{m'}(a) = f_m(a))\big)$$

that defines a check for address equality for all of the lvalues in the same region as pointer p (locations in the $Alloc(f)$ region are modeled by the uninterpreted function f, $m = Index(f)$ is a current version of f and $m' = m + 1$ is a new version).

By $\omega(s, f_i)$ we will understand a constant offset of a field f_i from the base address of struct type variable s. Because we assume that there is only one structure type `struct s` in our programs, $\omega(s, f_i)$ can be made just $\omega(f_i)$.

In B&B memory model implemented on top of CPAchecker's existing memory model the operator of a strongest post-condition is defined as $SP_{op}(\varphi) = \varphi \wedge \Gamma(op)$, where φ is a symbolic abstract state and constraints $\Gamma(op)$ are defined by table I.

C. Example

The following program will be considered correct if we use either of the memory models. Γ constraints in terms of B&B memory model for the program are shown in table II. Path formula can be made as conjunction of all formulas in Γ column of the table II. It is *unsat* in terms of either of the memory models. This means that the tool can not go by this path (i.e. won't consider it as a potential error trace candidate).

```
struct s { int f1, f2; };
struct s * p1;
struct s * p2;

p1 = alloc();
p2 = alloc();

p1 -> f1 = 6;
p2 -> f2 = 5;
assume(p1 -> f1 == p2 -> f2);
```

Why the conjunction is *unsat*?

1) In the existing memory model memory allocated for pointers $p1$ and $p2$ cannot intersect because it was allocated using $alloc()$ function (the corresponding path formula is not given).
2) In the given Γ constraints for this path (using the B&B model) the following contradicting elements are present:

Table I
Γ CONSTRAINTS CREATION RULES

Operation (op)	$Index$	$Alloc$	Base address index k'	Γ constraints
Variable allocation on stack struct s * p;	No changes	A' - new variable; $Alloc'(G) = \{A'\} \cup Alloc(G)$	k' - new index	p = $A' \wedge A' > 0 \wedge B(A') = k'$
Heap variable allocation p = alloc();	l' - new index for G, $l = Index(G)$, $Index' = Index \setminus \{G \to l\} \cup \{G \to l'\}$	A', A'_i - new variables $Alloc'(G) = \{A\} \cup Alloc(G)$ $Alloc'(F^i) = \{A'_i\} \cup Alloc(F^i)$, $i = \overline{1,n}$	k', k'_i - new indices, $i = \overline{1,n}$	$G_{l'}(p) = A' \wedge A' > 0 \wedge B(A') = k'$ $\wedge\, mem\_upd(p, G, l', l)$ $\bigwedge_{i=\overline{1,n}} \Big((G_{l'}(p) + \omega(f_i)) = A'_i$ $\wedge\, A'_i > 0 \wedge B^i(A'_i) = k'_i \Big)$
p = $undef\_ptr()$	l' - new index for G, $l = Index(G)$, m' - new index for $undef\_ptr$, $m = Index(undef\_ptr)$, $Index' = Index \setminus \big(\{G \to l\} \cup \{undef\_ptr \to m\}\big) \cup \{G \to l'\} \cup \{undef\_ptr \to m'\}$	No changes	No changes	$G_{l'}(p) = undef\_ptr_m \wedge mem\_upd(p, G, l', l)$
p $\to f_i$ = e	m' - new index for F^i, $m = Index(F^i)$, $Index' = Index \setminus \{F^i \to m\} \cup \{F^i \to m'\}$	No changes	No changes	$F^i_{m'}(G_l(p) + \omega(p, f_i)) = \Gamma(e)$ $\wedge\, mem\_upd(G_l(p) + \omega(f_i), F^i, m', m)$, where $l = Index(G)$ and $\Gamma(e)$ can be computed using the following rules: $\Gamma(const) : const$; $\Gamma(p2 \to f_j) : F^j_k(G_l(p2) + \omega(f_j))$, where $k = Index(F^j), l = Index(G)$ $\Gamma(e_1\ op\ e_2)$, $op \in \{'+', '-', '*', '/'\}$: $\Gamma(e_1)\ op\ \Gamma(e_2)$.
assume(p)	No changes	No changes	No changes	$\Gamma(p)$ for predicate p computes as following: $\Gamma(const) : const$; $\Gamma(s) : G_l(s)$, where $l = Index(G)$; $\Gamma(s \to f_i) : F^i_m(G_l(s) + \omega(f_i))$, where $m = Index(F^i), l = Index(G)$; $\Gamma(p1 == p2) : \Gamma(p1) = \Gamma(p2)$; $\Gamma(p1 < p2) : \Gamma(p1) < \Gamma(p2)$; $\Gamma(p1 <= p2) : \Gamma(p1) \leq \Gamma(p2)$; $\Gamma(p1 \Vert p2) : \Gamma(p1) \vee \Gamma(p2)$; $\Gamma(p1 \&\& p2) : \Gamma(p1) \wedge \Gamma(p2)$; $\Gamma(!p) : \neg\Gamma(p)$.

- $F^1_2(G_3(p1) + \omega(f1)) = F^2_2(G_3(p2) + \omega(f2))$;
- $F^2_2(G_3(p2) + \omega(f2)) = 5$;
- $F^1_2(G_3(p1) + \omega(f1)) = 6$.

Let's take a look at the example program below. In the program's source code there are calls to the function $undef\_ptr()$ that returns an unknown pointer. The pointer $p2$ is initialized using this function. Γ constraints in terms of B&B memory model for the program are shown in table III. Path formula can be made as conjunction of all formulas in Γ column of the table III.

```
void * undef_ptr();

struct s { int f1, f2; };
struct s * p1;
struct s * p2;

p1 = alloc();
p2 = undef_ptr();

p1 -> f1 = 6;
p2 -> f2 = 5;
assume(p1 -> f1 == p2 -> f2);
```

In B&B memory model $p1 \to f1$ and $p2 \to f2$ exist in the separate memory regions. In Γ constraints for this path the same contradicting elements as for the previous example are present. Thus the update of one of them wouldn't affect the other one. Because of that the result of verification would be that the error state is unreachable (path formula is still *unsat*).

However, in the existing memory model fields $f1$ and $f2$ of struct s exist in the same memory region and it uses only one uninterpreted function for them (see table 2 in [4]). Memory for their base pointers $p1$ and $p2$ was allocated using known $alloc()$ function and function $undef\_ptr()$ returning unknown pointer respectively. It cannot be confirmed that an update to a field $f2$ of the $p2$ wouldn't affect the access to the $f1$ struct field of $p1$. In the formula the location for field $f2$ of the $p2$ is $(G_3(p2) + \omega(f2))$ which is $undef\_ptr_1 + \omega(f2)$. The locations $(G_3(p1)+\omega(f1))$ and $(G_3(p2)+\omega(f2))$ exist in the same region and may be equal. Thus the formula is satisfiable. It means that the result of verification with existing memory model will be a reachable path to the program's error state.

Usually such situations in practice are false alarms because different fields of different structures do not normally intersect. Thus the assumptions related to this behavior in the existing memory model aren't really incorrect but they are quite improbable in practice. Usage of the B&B memory model will be able to reduce the number of false alarms caused by these assumptions (continued in section VI).

V. IMPLEMENTATION NOTES

The creation of memory regions is an automated process. In CPAchecker verification tool CFA (control-flow automaton) is used as an inner representation of the program. It is sufficient

Table II
EXAMPLE BUILD OF PATH FORMULA FOR THE CORRECT PROGRAM

Path instruction	$Index$	$Alloc$	k'	Γ
struct s * p1	$\{G \to 1, F^1 \to 1, F^2 \to 1\}$	$Alloc(G) = \{A_1\}$	1	$p1 = A_1 \wedge A_1 > 0 \wedge B(A_1) = 1$
struct s * p2	$\{G \to 1, F^1 \to 1, F^2 \to 1\}$	$Alloc(G) = \{A_1, A_2\}$	2	$p2 = A_2 \wedge A_2 > 0 \wedge B(A_2) = 2$
p1 = $alloc()$	$\{G \to 2, F^1 \to 1, F^2 \to 1\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	3, 4, 5	$G_2(p1) = A_3 \wedge A_3 > 0 \wedge B(A_3) = 3$ $\wedge (G_2(p1) + \omega(f1)) = A_4$ $\wedge A_4 > 0 \wedge B^1(A_4) = 4$ $\wedge (G_2(p1) + \omega(f2)) = A_5$ $\wedge A_5 > 0 \wedge B^2(A_5) = 5$ $\wedge mem\_upd(p1, G, 2, 1)$
p2 = $alloc()$	$\{G \to 3, F^1 \to 1, F^2 \to 1\}$	$Alloc(G) =$ $\{A_1, A_2, A_3, A_6\}$; $Alloc(F^1) = \{A_4, A_7\}$; $Alloc(F^2) = \{A_5, A_8\}$.	6, 7, 8	$G_3(p2) = A_6 \wedge A_6 > 0 \wedge B(A_6) = 6$ $\wedge (G_3(p2) + \omega(f_1)) = A_7$ $\wedge A_7 > 0 \wedge B^1(A_7) = 7$ $\wedge (G_3(p2) + \omega(f_2)) = A_8$ $\wedge A_8 > 0 \wedge B^2(A_8) = 8$ $\wedge mem\_upd(p2, G, 3, 2)$
p1 $\to$ f1 = 6	$\{G \to 3, F^1 \to 2, F^2 \to 1\}$	$Alloc(G) =$ $\{A_1, A_2, A_3, A_6\}$; $Alloc(F^1) = \{A_4, A_7\}$; $Alloc(F^2) = \{A_5, A_8\}$.	8	$F_2^1(G_3(p1) + \omega(p1, f1)) = 6$ $\wedge mem\_upd(G_3(p1) + \omega(f1), F^1, 2, 1)$
p2 $\to$ f2 = 5	$\{G \to 3, F^1 \to 2, F^2 \to 2\}$	$Alloc(G) =$ $\{A_1, A_2, A_3, A_6\}$; $Alloc(F^1) = \{A_4, A_7\}$; $Alloc(F^2) = \{A_5, A_8\}$.	8	$F_2^2(G_3(p2) + \omega(f2))) = 5$ $\wedge mem\_upd(G_3(p2) + \omega(f2), F^2, 2, 1)$
assume(p1 $\to$ f1 == p2 $\to$ f2)	$\{G \to 3, F^1 \to 2, F^2 \to 2\}$	$Alloc(G) =$ $\{A_1, A_2, A_3, A_6\}$; $Alloc(F^1) = \{A_4, A_7\}$; $Alloc(F^2) = \{A_5, A_8\}$.	8	$F_2^1(G_3(p1) + \omega(f1)) = F_2^2(G_3(p2) + \omega(f2))$

Table III
EXAMPLE BUILD OF PATH FORMULA FOR THE PROGRAM WITH UNKNOWN MEMORY FUNCTION

Path instruction	$Index$	$Alloc$	k'	Γ
struct s * p1	$\{G \to 1, F^1 \to 1, F^2 \to 1,$ $undef\_ptr \to 1\}$	$Alloc(G) = \{A_1\}$	1	$p1 = A_1 \wedge A_1 > 0 \wedge B(A_1) = 1$
struct s * p2	$\{G \to 1, F^1 \to 1, F^2 \to 1,$ $undef\_ptr \to 1\}$	$Alloc(G) = \{A_1, A_2\}$	2	$p2 = A_2 \wedge A_2 > 0 \wedge B(A_2) = 2$
p1 = $alloc()$	$\{G \to 2, F^1 \to 1, F^2 \to 1,$ $undef\_ptr \to 1\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	3, 4, 5	$G_2(p1) = A_3 \wedge A_3 > 0 \wedge B(A_3) = 3$ $\wedge mem\_upd(p1, G, 2, 1)$ $\wedge (G_2(p1) + \omega(p1, f_1)) = A_4$ $\wedge A_4 > 0 \wedge B^1(A_4) = 4$ $\wedge (G_2(p1) + \omega(f_2)) = A_5$ $\wedge A_5 > 0 \wedge B^2(A_5) = 5$
p2 = $undef\_ptr()$	$\{G \to 3, F^1 \to 1, F^2 \to 1,$ $undef\_ptr \to 2\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	5	$G_3(p2) = undef\_ptr_1$ $\wedge mem\_upd(p2, G, 3, 2)$
p1 $\to$ f1 = 6	$\{G \to 3, F^1 \to 2, F^2 \to 1,$ $undef\_ptr \to 2\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	5	$F_4^1(G_3(p1) + \omega(f1)) = 6$ $\wedge mem\_upd(F_2^1(G_3(p1) + \omega(f1)), F^1, 2, 1)$
p2 $\to$ f2 = 5	$\{G \to 3, F^1 \to 2, F^2 \to 2,$ $undef\_ptr \to 2\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	5	$F_2^2(G_3(p2) + \omega(f2)) = 5$ $\wedge mem\_upd(G_3(p2) + \omega(f2), F^2, 2, 1)$
assume(p1 $\to$ f1 == p2 $\to$ f2)	$\{G \to 3, F^1 \to 2, F^2 \to 2,$ $undef\_ptr \to 2\}$	$Alloc(G) = \{A_1, A_2, A_3\}$; $Alloc(F^1) = \{A_4\}$; $Alloc(F^2) = \{A_5\}$.	5	$F_2^1(G_3(p1) + \omega(f1)) = F_2^2(G_3(p2) + \omega(f2))$

to go through it and find in it all of the struct field accesses. This allows to distinguish those fields that don't have their address taken somewhere in the program.

In the implementation we do not take into consideration the possibility of field accesses through pointer arithmetic and through the usage of pointer conversions because of the high improbability of such field accesses in program's source code.

VI. EXPERIMENTS

To determine the efficiency of B&B memory model implementation in comparison to existing memory model of the CPAchecker tool a number of launches were performed on the predefined sets of Linux kernel modules. To use the implemented memory model one must have:

1) CPAchecker verification tool with revision number 23271 or higher from the branch *trunk*;
2) option cpa.predicate.useMemoryRegions should be set to 'true'.

A. False alarm set

Several of those were found during the verification of Linux kernel 3.14. The review of error traces allowed to determine situations when reachability of error state was present due to updates to same-typed pointers' memory. Those 26 kernel modules that caused false alarms due to pointer updates are included in this set. The goal of this experiment was to find out what effect the usage of B&B memory model will produce on the tools precision. Tables IV and V hold information about changes of the tool's verdicts.

Table IV
B&B APPLICABILITY

	B&B could help	B&B couldn't help
B&B helped	10	0
B&B didn't help	0	16

Table V
VERDICT CHANGES

False alarm → Safe	False alarm → Unknown	False alarm → False alarm*
3	5	2

* - different error trace and cause of unsafe

B. Linux 4.2-rc1 kernel modules

A set of Linux kernel drives (version 4.2-rc1) was selected to study the efficiency of B&B memory model implementation in comparison to the existing memory model of the CPAchecker tool.

The launch was performed for rule that checks correctness of functions working with usb_get_* and usb_put_* functions of usb-system. Launch results can be found in tables VI, VII. Launch configuration:

- time limit – 15 minutes;
- memory limit – 15 Gb;
- number of CPU cores – 4;

The differences in the regions the models have lead to the difference in program's paths that are covered by the tool. This explains Unsafe → Unknown, Unknown → Safe and Unknown → Unsafe transitions, where *Safe* means that program's error state is unreachable, *Unsafe* – error state is reachable, *Unknown* - timeout or runtime error. This experiment's results show that the improvement to the tool's precision is present while the verification speed is competitive.

Table VI
LINUX 4.2-RC1 STATISTICS

	Existing model	B&B
Verification time	35.8 hours	35.3 hours
Safe	4245	4241
Unsafe	69	68
Unknown	161	166

Table VII
TRANSITIONS

Existing model \ B&B	Safe	Unsafe	Unknown
Safe	4240	0	5
Unsafe	0	67	2
Unknown	1	1	159

C. SV-COMP'17 DeviceDrivers64

This set contains files from the DeviceDrivers64 set of the international competition on software verification SV-COMP'17. It consists of 2795 modules of different Linux kernel versions. Launch configuration:

- time limit – 15 minutes;
- memory limit – 15 Gb;
- number of CPU cores – 4;

Several of the transitions from the incorrect results can be explained by the difference in models choice of pointer's may-aliases. The same modules were present in earlier mentioned *False alarm* set. Several transitions to Unknown can be explained by the addition of overhead costs required for B&B usage to the verification tasks on the verge of timeout. It is worth mentioning that usage of B&B memory model reduced time for SMT-solver by roughly 8% for the correct results.

Table VIII
DEVICEDRIVERS64 STATISTICS

Memory models	Existing	B&B
Total number of files	2795	2795
Correct results	1791	1780
Error state unreachable	1524	1522
Error state reachable	267	258
Incorrect results	7	5
Missed errors	4	4
False alarm	3	1
Unknown	1004	1015

Table IX
TIME FOR DEVICEDRIVERS64 SET

Memory models	Existing	B&B
Total time	143.6 hours	143.1 hours
Time for correct results	14.9 hours	14.1 hours
SMT solver time	10500 sec (2.9 hours)	12400 sec (3.4 hours)
SMT solver time for correct results	660 sec	605 sec

Table X
VERDICT CHANGES

Transitions from Existing model (EM) to B&B model	Quantity
Correct result (EM) → Incorrect result	0
Correct result (EM) → Unknown	16
Incorrect result (EM) → Correct result	2
Incorrect result (EM) → Unknown	0
Unknown (EM) → Correct result	3
Unknown (EM) → Incorrect result	0

VII. CONCLUSION

This paper proposes the specification of B&B memory model and its region-based reasoning in terms of uninterpreted functions. Its implementation on top of existing memory model of the CPAchecker verification tool provides better verification precision while the verification speed remains competitive. The implementation was included in the official repository of the CPAchecker static verification tool.

REFERENCES

[1] V. Kuliamin, "Metody verifikatsii programmnogo obespecheniya [Software verification methods]," *Vserossiiskii konkursnyi otbor obzorno-analiticheskikh statei po prioritetnomu napravleniyu "Informatsionno-telekommunikatsionnye sistemy" [Russian national competitive selection of review and analytical articles in priority direction "Information and telecommunication systems"]*, p. 117, 2008 (in Russian).

[2] D. Beyer, T. A. Henzinger, and G. Theoduloz, "Configurable Software Verification: Concretizing the Convergence of Model Checking and Program Analysis," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, W. Damm and H. Hermanns, Eds., vol. 4590. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 504–518.

[3] M. Mandrykin and A. Khoroshilov, "A memory model for deductively verifying Linux kernel modules," *A.P. Ershov Informatics Conference, the PSI Conference Series, 11th edition*, 2017 (to appear).

[4] M. Mandrykin and V. Mutilin, "Modelirovanie pamyati s ispol'zovaniem neinterpretiruemykh funktsii v predikatnykh abstraktsiyakh [Modeling Memory with Uninterpreted Functions for Predicate Abstractions]," *Trudy ISP RAN [Proceedings of the Institute for System Programming]*, vol. 27, pp. 117–142, 2015 (in Russian).

[5] R. Bornat, "Proving pointer programs in Hoare Logic," in *Mathematics of Program Construction: 5th International Conference, MPC 2000*, ser. Lecture Notes in Computer Science, R. Backhouse and J. N. Oliveira, Eds., vol. 1837. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 102–126.

[6] R. Burstall, "Some techniques for proving correctness of programs which alter data structures," *Machine Intelligence*, vol. 7, pp. 23–50, 1972.

Static Verification of Linux Kernel Configurations[(1)]

Svyatoslav Kozin[#1] Vadim Mutilin[#2]

[#1]Software Engineering Department, HSE

Russia, Moscow, Myasnitskaya 20, 101000

[1]`kozyyy@yandex.ru`

[#2]ISP RAS

Russia, Moscow, Alexander Solzhenitsyn st., 25, 109004

[2]`mutilin@ispras.ru`

Abstract— **The Linux kernel is often used as a real world case study to demonstrate novel software product line engineering research methods. To provide the most safe experience of building of Linux product line variants it is necessary to analyse Kconfig file as well as source code. Tens of thousands of variable statements and options even by the standards of modern software development. Verification researchers offered lots of solutions for this problem. Standard procedures of code verification are not acceptable here due to time of execution and coverage of all configurations, so it was offered to check the operating system with special wrapper for tools analyzing built code and configuration file. Such a bundle is able to provide efficient tool for calculating all valid configurations for predetermined set of code and Kconfig. These analyses can be used for improving existing analysis tools as well as decision of choice the right configuration. Our main goal is to contribute to a better understanding of possible defects and offer fast and safe solution to improve the validity of evaluations based on Linux.**

Keywords— Software Product Lines, Linux, Kconfig, Preprocessor.

I. Introduction

Nowadays, software is used to solve increasingly important and complex tasks, due to this fact the complexity of software architectures is also constantly growing. With the increasing complexity of programs, the complexity of development, analysis and maintenance arises. There are many methods that allow you to reduce the costs of supporting the software life cycle. One such method is the creation of variable systems (or family of systems, software product families, software product lines) [1-4]. The superiority over the usual development method is that systems are manufactured with the condition of multiple elements used for several systems with a similar set of functions, taking into account a specific target audience of users. At the same time, a complex and widely known representative of variable software is the Linux operating system.

In the development of variable systems, a variability model and a variation mechanism play a fundamental role. The variability model specifies the space of possible variants of this family of systems. Usually it is determined by a set of features or configuration parameters, by the sets of their possible values and constraints on possible combinations of these values, each variant of the system corresponds to a certain set of values of all features. The variation mechanism provides the ability to build all possible system variants from a limited set of created and followed artifacts. In Linux, the variability model and its relationship to the variation mechanism is built on the basis of Kconfig files, Makefile files and additional scripts. Kconfig describes all possible features, as well as their relationship with each other in a special language. Then on the basis of Kconfig, the configuration file .config is defined, which describes the version of the system. It consists of a set of configuration variables described in Kconfig and values that satisfy the constraints of Kconfig. During kernel assembly, the values of variables specified in .config are passed to the code as constants for the preprocessor and to Makefiles, which will be used in the variation mechanism [5].

In the field of operating systems (hereinafter the operating system we mean the kernel and the

(1) Supported by grant RFBR 16-01-00352 A

underlying OS libraries, providing the interfaces for work with computing resources and hardware), a mechanism of conditional compilation of C / C ++ languages is widely used as a mechanism for variability of the mixed type (based on macros #ifdef, #else). It allows you to compose code at the build stage that combines various variables specified by a set of characteristic values that are conditional compilation parameters in this case (defined by the #define and #undef macros, as well as preprocessor setup parameters).

The complexity of variability models for modern operating systems is very high, for example, the Linux kernel version 2.6.32 has 6319 characteristics, more than 10,000 constraints that can be used up to 22 individual characteristics, with the majority of characteristics depending on at least 4 others, and the maximum depth of the dependency tree is 8 [6]. This complexity causes a large number of errors, primarily due to the difficulty of taking into account all the factors that a developer of a separate code element should do. To identify and cope with these errors, it is necessary to use specialized techniques of analysis and verification. Complexity of analysis that is typical for systems with such a variability mechanism arise because of the huge size of the possible variants space (which makes it completely unrealistic to check them all). Due to using of conditional compilation, each fragment does not have to be a separate component with a certain behavior that can be analyzed separately from the rest of the code, usually such fragments are just insertions into the common code, and can only be checked in certain combinations with each other. The need to solve these problems imposes special requirements on the tools and analysis methods used for complex variable operating systems and system software in general. These requirements are specific for analysis and verification - the methods used to create such systems, by themselves, do not facilitate their analysis [7,8]. The main goal of this work is to propose a methodcapable of coping with the verification of the Linux OS taking into account the variability, to give acceptable accuracy of verification and speed of execution comparable to the verification of common programs.

An errors analysis of such complex systems as Linux can be done with a lot of different tools. The most convenient of them are static analyzers and static verifiers. Static code analysis is the analysis of software produced (as opposed to dynamic analysis) without real execution of the programs under investigation. Existing static analyzers (such as Coverity[9] or Svace[10]) and static verifiers (such as BLAST[11], CPAchecker[11]) are only designed to work with the code already compiled. Accurate analysis requires pre-assembling of a specific configuration, and only after that start of the actual analysis. As a result, the total inspection time becomes unacceptably large. There is a class of tools that are focused on analysis of a set of possible configurations, they do not split the phase of building configurations and code analysis. As the example of such tools we can take a look at TypeChef and Undertaker. These tools are designed to solve special problems in the sphere of variability. Undertaker is looking for a "dead code" - such a problems when different configurations give the same product, besides it has a lot of built-in helper modules to provide main task, and one more function - assembling the minimal Linux kernels for individual use cases. TypeChef is looking for linking and compile errors with a variability-aware method. It is important to say, that TypeChef can not find difficult problems in code like complex static analysers. Suggested in this paper tool should analyze code deeply like BLAST or CPAchecker and, on the other hand, should do it in variability-aware way without all-configuration brute forcing. For this task we suggest to use CPAchecker due to its outstanding abilities in the error findings, despite CPAchecker’s expenditure of time[11]. The maximum reduction in verification time should be achieved through the optimal selection of configurations that will be directed to the input of the static analyzer.

II. Configuration Set Selection

The problem of selecting configurations can be considered as a splitting of the configuration space into equivalence classes. Each of the selected configurations will belong to one of the classes. To split the configuration space into classes, it is suggested to use the MC/DC test coverage criterion. The advantage of this choice is that it allows you to significantly reduce the number of classes, even for large software systems. In addition, logic of analysis/construction of configurations in Kconfig representation is similar to logic of control flow graph analysis of common programs, where the MC/DC criterion has proven itself well.

A set is considered to reach 100% coverage by this metric, if:

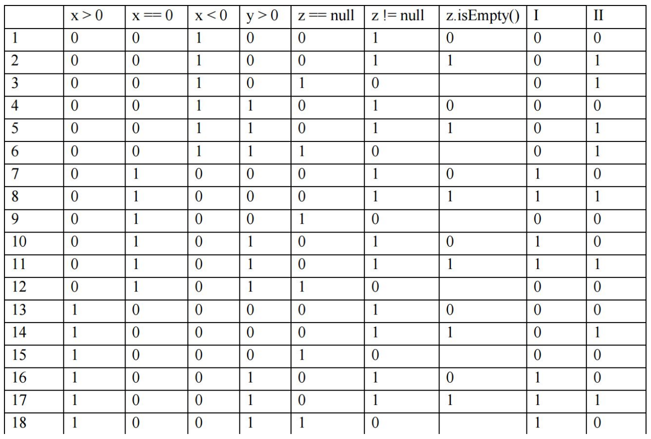
- Each entry and exit point is invoked
- Each decision takes every possible outcome
- Each condition in a decision takes every possible outcome
- Each condition in a decision is shown to independently affect the outcome of the decision.

In other words, in the full test, in accordance with the MC/DC coverage criterion, it should be demonstrated that every elementary condition that can influence the resulting value of the branching condition that includes it actually changes its value regardless of the other elementary conditions.

Example:

Our program has next 2 conditions of branching: *(x > 0) && (y > 0) || (x == 0) && (z != null) and (a > 0) && (z == null) || (z != null) && (z == IS_EMPTY)* where ‘x’, ‘y’, ‘z’, ‘a’ are some config variables, IS_EMPTY - is a macros (hereinafter will be z.isEmpty() for the convenient perception). Also as example, suppose ‘a = !x’ in the model of Kconfig file. So we can transform this formula in *(x > 0) && (y > 0) || (x == 0) && (z != null) and (x < 0) && (z == null) || (z != null) && z.isEmpty()*. We have just got rid from the ‘a’ variable, replacing it with equivalent ‘x’ expression.

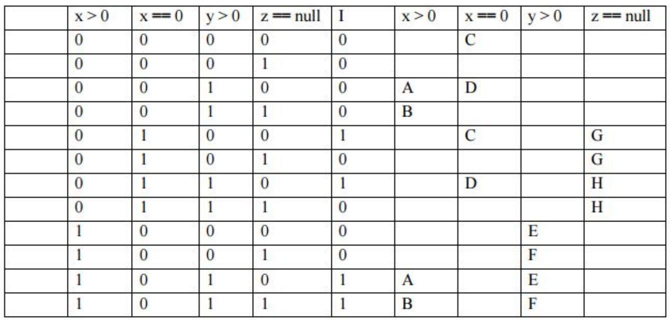
Let’s construct table (Picture 1) for all available combinations of elementary conditions, keeping in mind that z.isEmpty() is defined if only z!=null is true.



	x > 0	x == 0	x < 0	y > 0	z == null	z != null	z.isEmpty()	I	II
1	0	0	1	0	0	1	0	0	0
2	0	0	1	0	0	1	1	0	1
3	0	0	1	0	1	0		0	1
4	0	0	1	1	0	1	0	0	0
5	0	0	1	1	0	1	1	0	1
6	0	0	1	1	1	0		0	1
7	0	1	0	0	0	1	0	1	0
8	0	1	0	0	0	1	1	1	1
9	0	1	0	0	1	0		0	0
10	0	1	0	1	0	1	0	1	0
11	0	1	0	1	0	1	1	1	1
12	0	1	0	1	1	0		0	0
13	1	0	0	0	0	1	0	0	0
14	1	0	0	0	0	1	1	0	1
15	1	0	0	0	1	0		0	0
16	1	0	0	1	0	1	0	1	0
17	1	0	0	1	0	1	1	1	1
18	1	0	0	1	1	0		1	0

Pic. 1

Next table represent only those combinations, that are essential for the first branching. Next, we will find all those pairs, where only one elementary condition is changed and all condition result is changed too (Picture 2).

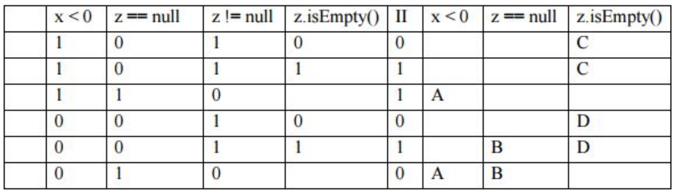


	x > 0	x == 0	y > 0	z == null	I	x > 0	x == 0	y > 0	z == null
	0	0	0	0	0		C		
	0	0	0	1	0				
	0	0	1	0	0	A	D		
	0	0	1	1	0	B			
	0	1	0	0	1		C		G
	0	1	0	1	0				G
	0	1	1	0	1		D		H
	0	1	1	1	0				H
	1	0	0	0	0			E	
	1	0	0	1	0			F	
	1	0	1	0	1	A		E	
	1	0	1	1	1	B		F	

Pic. 2

Relevant combinations are marked in last 4 columns of the table. Two rows are marked with one letter, if they form a pair of combinations. This letter is placed in a column according to elementary condition, that affected on a result. To provide full coverage with MC/DC for this branching we have to select pairs of combinations, matching at least one letter in each of the last four columns. It is enough to select combinations marked as A, D, E, H (5 combinations) or B, C, F, G (6 combinations).

There is a similar table for the second branch condition below (Picture 3). Conditions (z == null) and (z! = Null) uniquely determine the values of each other, so you can consider only one of them. In addition, you can not change the value (z == null) and do not change the values of z.isEmpty () - this condition is computable only for one (z==null) result.

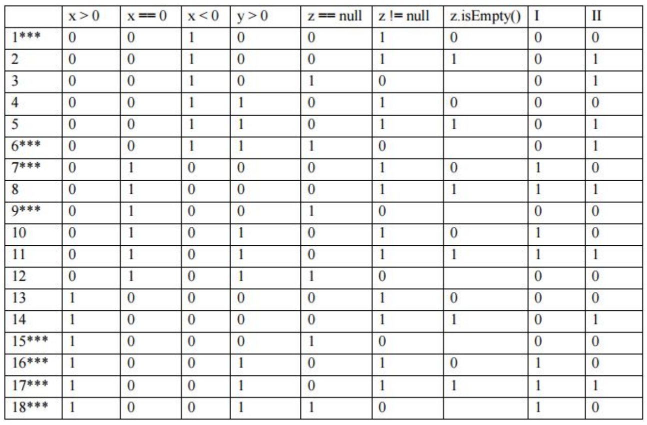


	x < 0	z == null	z != null	z.isEmpty()	II	x < 0	z == null	z.isEmpty()
	1	0	1	0	0			C
	1	0	1	1	1			C
	1	1	0		1	A		
	0	0	1	0	0			D
	0	0	1	1	1		B	D
	0	1	0		0	A	B	

Pic. 3

In this way, it is possible to select set of combinations marked as A,B,D (4 combinations).

Final set of combinations for MC/DC metric marked with '***' in a table below (Picture 4).



	x > 0	x == 0	x < 0	y > 0	z == null	z != null	z.isEmpty()	I	II
1***	0	0	1	0	0	1	0	0	0
2	0	0	1	0	0	1	1	0	1
3	0	0	1	0	1	0		0	1
4	0	0	1	1	0	1	0	0	0
5	0	0	1	1	0	1	1	0	1
6***	0	0	1	1	1	0		0	1
7***	0	1	0	0	0	1	0	1	0
8	0	1	0	0	0	1	1	1	1
9***	0	1	0	0	1	0		0	0
10	0	1	0	1	0	1	0	1	0
11	0	1	0	1	0	1	1	1	1
12	0	1	0	1	1	0		0	0
13	1	0	0	0	0	1	0	0	0
14	1	0	0	0	0	1	1	0	1
15***	1	0	0	0	1	0		0	0
16***	1	0	0	1	0	1	0	1	0
17***	1	0	0	1	0	1	1	1	1
18***	1	0	0	1	1	0		1	0

Pic. 4

We can notice that from 18 possible combinations, we can use only 8 to get full coverage according to MC/DC method.

In general, the MC / DC metric allows 2n different situations to be used instead of 2n condition combinations.

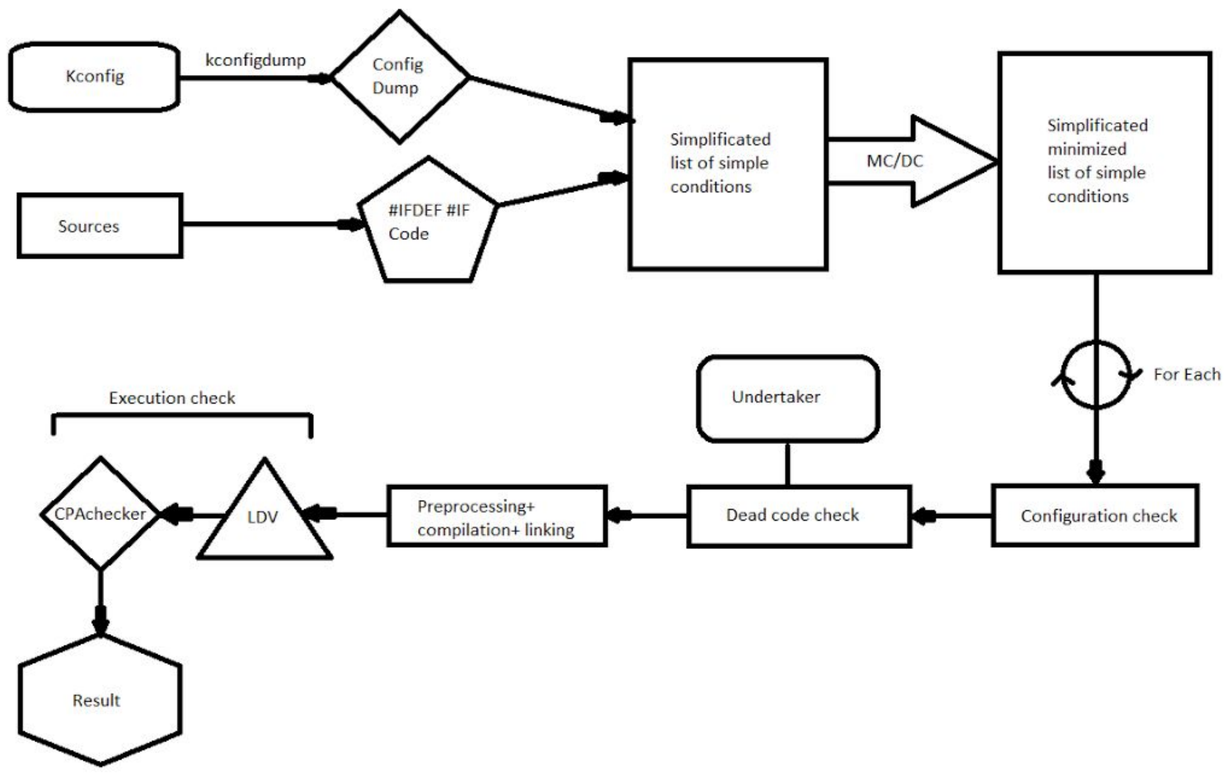
III. Kernel Check Stages

For each of the found configurations, you should run a kernel verification. The program will scan the kernel in 6 stages: configuration, search for a "dead code", preprocessing, compiling, linking, searching for run-time errors. It is also worth noting that we will look for not only errors, but also configuration defects. Defects - is a broader concept, and it includes not only system errors, but also possible errors of the kernel without processing the interrupt.

Description of the stages:

A. Configurations.
   At the time of build, Linux itself checks the configuration file, but it's worth checking it for recursive dependencies and non-existent variables in the code, but existing in Kconfig (and vice versa).
B. Search for a "dead code".
   A "dead" code is such a code in which control is not transferred under any circumstances. This code contributes to a common configuration error when two different configurations produce the same product at the output. This problem can be solved using the built-in tools of the Undertaker.
C. Preprocessing.
   Preprocessing is performed just before compilation and at this stage we will get the code that will be compiled into the final version of the program. Most errors at this stage can find a preprocessor. It remains for us to inform the user of a possible conflict of names if we see duplicate names of preprocessor variables and their redefinition, because they have one nominal space and the developer may not notice the problem of overriding.
D. Compilation.
   Compilation is complete on the compiler side. Here there are such errors as: detection of an undeclared variable / function, missing punctuation in the code, etc.
E. Linking.
   As well as compilation is completely redirected to the linker. The linker finds errors in missing libraries or files.
F. Execution Errors.
   The most difficult part of the test is using the LDV and CPAchecker tools. LDV assigns labels to the code of the program according to the preset rules, while CPAchecker searches for them and builds accessibility graphs to them so we can see how a label is achieved.

Picture below is a visual representation of the program:



Conclusion

This article describes the method, which allows you to check the Linux operating system for errors without being depended on the configuration. This approach provides high code coverage and an improved speed of verification comparing to brute force method. This software product can be used in the production of distributions, as well as for verification of existing ones.

References

[1] Jacobson I., Griss M., Jonsson P. Software Reuse, Architecture, Process and Organization for Business Success. Addison-Wesley, 1997. ISBN-13: 978-0201924763.

[2] Bosch J. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach. Pearson Education, 2000. ISBN-13: 978-0201674941.

[3] Clements P., Northrop L. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering, Addison-Wesley, 2001. ISBN-13: 978-0201703320.

[4] Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.

[5] Kuliamin V.V., Lavrischeva E.M., Mutilin V.S., Petrenko A.K. “Верификация и анализ вариабельных операционных систем” Труды Института системного программирования РАН, vol 28, 2016, pp. 189-208.

[6] Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the Linux kernel variability model. Proc. of SPLC’10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.

[7] Lavrischeva E. M., Koval G.I., Slabospickaya O.O., Kolesnik A.L. Особенности процессов управления при создании семейств программных систем. Проблемы программирования, (3):40-49, 2009.

[8] Lavrischeva E. M., Koval G.I., Slabospickaya O.O., Kolesnik A.L. Теоретические аспекты управления вариабельностью в семействах программных систем. Весник КНУ, серия физ.–мат. наук, (1):151-158, 2011.

[9] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler, A “Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World”, Communications of the ACM, Vol. 53 No. 2, pp. 66-75

[10] Borodin A.E., Belevancev A.A, “Статический анализатор Svace как коллекция анализаторов разных уровней сложности” Труды ИСП РАН, vol 27, No. 6, 2015., pp. 111-134

[11] Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, “The software model checker BLAST”, Int J Softw Tools Technol Transfer (2007) 9:505–525, Springer-Verlag 2007

[12]

A Technique for Parameterized Verification of Cache Coherence Protocols

V.S. Burenkov[1]

[1]JSC MCST, burenkov_v@mcst.ru

Abstract**—This paper introduces a technique for scalable functional verification of cache coherence protocols that is based on the verification method, which was previously developed by the author. Scalability means that verification efforts do not depend on the model size (i.e. the number of processors in the system under verification). The article presents an approach to the development of formal Promela models of cache coherence protocols and shows examples taken from the Elbrus-4C protocol model. The resulting formal models consist of language constructs that directly reflect the way protocol designers describe their developments. The paper describes the development of the tool, which is written in the C++ language with the Boost.Spirit library as parser generator and automatically performs the syntactical transformations of Promela models. These transformations are part of the verification method. The procedure for refinement of the transformed models is presented. Finally, the overall verification technique is described. The technique has been successfully applied to verification of the MOSI protocol implemented in the Elbrus computer systems. Experimental results show that computer memory requirements for parameterized verification are negligible and the amount of manual work needed is acceptable.***

Keywords**—multicore microprocessors, shared memory multiprocessors, cache coherence protocols, model checking, Spin, Promela.***

I. Introduction

Shared memory multiprocessors constitute one of the most common classes of high-performance computer systems. In particular, *multicore microprocessors*, which combine several processors (cores) on a chip, are widely used [1]. The number of cores is constantly increasing. The presence of cache memories that are local to each core determines the need for ensuring coherent memory state. To satisfy the need, microprocessor developers design and implement in hardware cache coherence protocols [2].

Cache coherence mechanisms are extremely complex. Therefore, both the design and their implementation are error-prone. Being especially critical, protocol bugs should be revealed before implementing the hardware. The widely recognized method for protocol verification is *model checking* [3]. It is fully automated, but suffers from a principal drawback – it is not scalable due to the *state space explosion* problem. Verification of a cache coherence protocol for five or more processors is impossible (at least, highly problematic) with the traditional methods [4].

To overcome the problem and develop scalable verification technologies, researchers focus mostly on verification of *parameterized designs* [3]. Previous articles of the author [5–8] presented a method for parameterized verification of cache coherence protocols. The author successfully applied the method to verification of the cache coherence protocol of the Elbrus-4C computing system. This paper presents an approach to the development of formal Promela models that can be analyzed by the verification method, describes the development of the tool that performs transformations of Promela models according to the method and presents the overall verification technique.

The paper is structured as follows. Section 2 takes a brief look at related work and provide the necessary links. Section 3 considers the question development of Promela models of cache coherence protocols. In Section 4, we describe how to perform parameterized verification of the Promela models in a semi-automatic way. We examine the development of the tool that automates parts of the verification method used. We present a technique for cache coherence protocols verification. Section 5 provides experimental results on using the technique for verifying the Elbrus-4C protocol. Section 6 summarizes the work and defines further research directions.

II. Related Work

This work extends the previous works [5–8] by dealing with the question of practical application of the method for parameterized verification of cache coherence protocols presented in those works.

Article [5] presents a review of related work and gives the motivation for development of a new method. The developed method is based upon works [9–13] that present a method of compositional model checking, which is based on syntactical transformations of models written in the Murφ language and counterexample-guided abstraction refinement.

The method [5–8] is used in the context of the following verification process:

1. Development of formal models of cache coherence protocols.
2. Parameterized verification by means of the method.

III. Development of Formal Models

It is highly desirable to have a modeling language that allows us to conveniently describe cache coherence protocols. To choose or develop such a language, we need to define a mathematical model of cache coherence protocols.

In accordance with the microprocessor system model that is used in work [2] for representation and analysis of cache coherence protocols, I chose to model cache coherence protocols as a set of communicating finite-state machines.

An element of this set may be either a cache controller or the system commutator. Let us define these notions. Each memory device of the microprocessor is operated by a *coherence controller*, which is a finite-state machine. Coherence controllers are coordinated by a special device – the *system commutator* – that is also a finite-state machine. A set of these machines constitutes a distributed system, in which the machines communicate by message passing in order to maintain cache coherence.

Each coherence controller connected with cache memory logically implements a set of independent and identical finite-state machines, one for each cache line. These machines are called *cache controllers*. Due to the independence and identity of cache controllers, it is customary to reflect only one cache line in the models of cache coherence protocols.

The states of cache controllers are divided into two classes: Stable states and transient states. Stable states of cache controllers are often the subset of the common set Modified, Owned, Exclusive, Shared, Invalid [2]. Transitions between these states are not atomic and occur through transient states. Transient states are specific to each microprocessor and their presence is one of the factors that determine high verification complexity.

Conditions that define correctness of cache coherence protocols are formulated as statements about stable states, for example: "Cache line can never be in Modified state in two caches simultaneously" [5]. Such statements belong to the class of invariant properties [14].

Usage of a set of communicating finite-state machines as the model of cache coherence protocols and invariant properties for specification defined the choice of the Promela language for modeling cache coherence protocols:

- In contrast to other languages (for example, Murφ and NuSMV), Promela provides process types and the means of synchronous and asynchronous interprocess communication (channels).

- Promela provides convenient specification language, which is Linear Temporal Logic (LTL).

- Spin – the system that implements Promela – provides different verification algorithms and optimizations, and is a modern and constantly developing tool.

The question of development of formal models of cache coherence protocols is insufficiently covered in the literature. Here, I present an approach to the construction of such models. According to the approach, a formal model of a cache coherence protocol of a system with n cores consists of n Promela processes for cache controllers and one Promela process for the system commutator.

For the considered cache coherence protocols, the following property holds: Only one initial request may be in process at a given point in time. System commutator performs a sequence of steps during the request processing, for example, the reception of the initial request and its analysis, sending of snoop- and other requests according to the results of the analysis, reception of the answers to these requests. Initial requests correspond to the memory access instructions that the processor core is executing. Reception of messages from other devices can only occur at particular steps. Thus, it is convenient to represent the system commutator as a Promela process whose body simply consists of operators that follow each other (Figure 1).

```
proctype **system_commutator**() {
again:
 <receive initial request>
 <analyze the initial request>
 <send coherent requests>
 <receive answers to coherent requests or the request completion message>
 <finalize the request processing>
goto again }
```

Figure 1 – Structure of the System Commutator Process

Cache controllers operate differently. On the one hand, we still may identify a number of steps, for example, sending an initial request, changing state from stable to transient, receiving snoop-requests. On the other hand, the relative order of these steps is often unspecified, and the same messages from other devices may be processed in different states of a cache controller. Thus, it is convenient to represent processes of this kind as infinite do-cycles consisting of the guarded commands (Figure 2).

```
proctype **cache_controller**() {
 do
 :: <send initial request from main states>
 :: <receive and process snoop-requests>
 :: <receive answers to coherent requests>
 :: <send the completion message>
od }
```

Figure 2 – Structure of Cache Controller Processes

See papers [5, 6, 8] for more details on how to organize processes and their communication.

For example, modeling of a situation in which cache controller sends an initial request and the system commutator receives it, may be performed as follows:

```
mtype cache[N] = I; // states of cache line
proctype cache_controller(byte i) {
do
:: atomic {cache[i] == I ->
// send initial request and change state
if :: ini_req_chan ! R, i; cache[i] = WR;
   :: ini_req_chan ! RI, i; cache[i] = WRI;
      ...
fi }
...
od }
```

```
proctype system_commutator(byte i) {
message_t message;
again:
// receive initial request
atomic {ini_req_chan ? message;
           curr_command = message.opcode;
           curr_client = message.requester;
}
if :: atomic {
// send snoop-request as a response
// to the initial request
curr_command == R ->
    coh_req_chan[0] ! snR, curr_client; ...
}
...
// receive acknowledgement
final_ack_chan ? message;
goto again; }
```

As another example, reception of a snoop-request by cache controller and generation of the response can be modeled as follows:

```
proctype cache_controller(byte i) {
do
...
:: atomic {nempty(coh_req_chan[i]) ->
    // receive snoop-request
    coh_req_chan[i] ? message;
if ...
   // analyze state...
:: cache[i] == WI_O
  // ... and the snoop-request type
  && message.opcode == snI ->
    // send corresponding answer
    coh_ans_chan ! ack, i;
    cache[i] = WRI;
... fi }
... od
}
```

Developers of cache coherence protocols describe and reason about their protocols in terms of message passing, and, as these examples show, their reasoning can be directly expressed in Promela. Moreover, the proposed organization of Promela processes allows verification engineers to perform quick changes that are needed to reflect the modifications of the cache coherence protocol under verification that occur in the course of its development.

IV. Parameterized Verification of Cache Coherence Protocols

The method for parameterized verification of cache coherence protocols presented in works [5, 6, 8] consists of two stages:

1. Performing the syntactical transformations of Promela models.

2. Refining the obtained model in accordance with the proposed procedure.

Model transformations have the following effect:

1. Reduction of the number of processes from $n+1$ (n cache controller processes and one system commutator process) to 4: two fully functioning cache controller processes, one abstract cache controller process that models the environment of the two processes, and the system commutator process. This transformation is possible due to the symmetry inherent in models of cache coherent protocols (all cache controller processes are identical and interchangeable, they do not have behaviors that depend on a particular process index value) and because the specification of cache coherence protocols only contains properties that regard the state of cache line in two caches.

2. Syntactical transformations of Promela operators constituting the model.

These transformations preserve invariant properties. This means that if such a property is true for the reduced model, then it is true for the initial model. A mathematical proof of the corresponding theorem is presented in articles [5, 6, 8].

A. *Performing the Syntactical Transformations*

The syntactical transformations presented in [5, 6, 8] may be performed manually. However, manual model modification is a very tedious, laborious and error-prone process. Moreover, some of the errors made may go undetected, as they will only lead to incorrect state space reduction and not to counterexamples. Therefore, it is highly desirable to perform the transformations automatically. To achieve that, I have developed a dedicated tool. With this tool, the verification engineer simply provides their Promela model as input to the tool, and the tool generates the transformed Promela model.

To automate the syntactical transformations, I have used a widespread approach to this kind of problems, according to which a tool builds the abstract syntax tree that represents the syntactical structure of the source code and then performs the transformations upon the tree traversal (Figure 3).

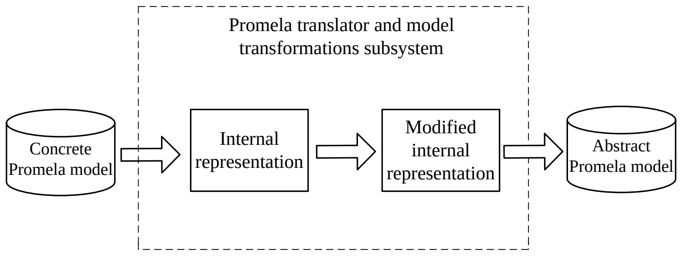

Promela translator and model transformations subsystem
Concrete Promela model
Internal representation
Modified internal representation
Abstract Promela model

Figure 3 – Scheme of Automated Model Transformation

Abstract syntax trees are usually constructed by parsers. There are two ways of parser implementation: manual and by means of a parser generator tool (for example, Bison, ANTLR, Boost.Spirit). Due to the unnecessary complexity of the first approach, I have chosen the second one.

The Boost.Spirit library was chosen as the parser generator, because:

- Boost.Spirit promotes modern usage of the C++ language that allows us to work with abstractions, which are suitable for a given domain, without performance loss.
- Boost.Spirit eliminates the need for additional tools like Bison or ANTLR: The only tools needed are a C++ compiler and the Boost library.

- The grammars that Boost.Spirit accepts are attributed, which results in a very convenient way of abstract syntax tree generation.
- Boost.Spirit contains a number of built-in parsers.
- The generated parsers are very efficient [15].

The mechanism of synthesized and inherited attributes allows us to simplify the task of abstract syntax tree generation by dividing it into two sequentially performed subtasks:

1. Development of the grammar, testing and debugging of the grammar. During this step, we only need to focus on the question of whether the grammar can correctly determine the syntactical correctness of a Promela model.

2. Development of data structures for the nodes of the abstract syntax tree and definition of the types of attributes of the grammar rules. The attribute mechanism allows Boost.Spirit to generate abstract syntax trees automatically, without any need for the addition of node construction operators to the grammar.

Usage of the abstract syntax tree generated by Boost.Spirit as an intermediate representation of Promela models allowed us to divide the task of performing the syntactical transformations automatically into three subtasks:

1. Development of Promela grammar in the C++ language by means of Boost.Spirit.

2. Development of data structures for abstract syntax tree representation.

3. Development of algorithms for abstract syntax tree traversal and abstract model generation.

Promela grammar is presented in [16]. Its implementation in C++ using Boost.Spirit looks similarly to that description. However, as Boost.Spirit generates recursive descent parsers, I have eliminated left recursion from the grammar.

Data structures for the nodes of abstract syntax tree are developed according to the information that we want the nodes to represent and attribute propagation rules defined in Boost.Spirit's documentation. In the developed tool, data structures that correspond to the synthesized attributes of the Promela grammar rules, contain information about nonterminals that are part of the rules. This is a very straightforward and convenient way of implementation of these data structures. For example, the following rule that describes the nonterminal "module" of the Promela grammar

```
qi::rule<Iter, module(), Skipper> module;
module =
    proctype
    | init
    | ltl
    | utype
    | mtype
    | decl_lst
    | ';' ;
```

has a synthesized attribute of type `module`, which is implemented as follows:

```
using module = boost::variant<
    proctype,
    init,
    ltl,
    utype,
    mtype,
    decl_lst
    >;
```

All the other nonterminals mentioned in this example have synthesized attributes of types implemented in a similar way.

The abstract syntax tree, which is generated automatically by Boost.Spirit based on the grammar and the attribute mechanism, consists of nodes of different types. Traversal of such tree is performed uniformly by means of visitors, as advocated by the Boost.Spirit documentation.

The syntactical transformations are performed during the abstract syntax tree traversal. I classified the transformations, most of which turned out to be in one of the three categories (transformations of assignments, transformations of expressions, transformations of communication actions), and precisely described them. To automatically carry them out, I have developed a number of abstract syntax tree modification algorithms and implemented them as part of the visitation mechanism. Printing out the modified syntax tree gives us the abstract Promela model.

For example, when generating the code for the abstract process, the following piece of Promela code

```
proctype cache_controller(byte i) {
do
...
:: (cache[i] == M_MAU || cache[i] == M_MAU_I)
&& (message.opcode == wb_ready) ->
    final_ack_chan ! data, i;
    cache[i] = I
```

is transformed into

```
proctype cache_controller_abs(byte i) {
do
...
:: true ->
    final_ack_chan ! data, i;
```

This example demonstrates the transformations of expressions and the assignment operator.

B. *Abstraction Refinement*

Execution of each type of initial requests consists of a particular sequence of events presented in the cache coherence protocol documentation. Considerations about the ordering of the events inspired the following refinement procedure:

1. For each type of initial requests define (according to the documentation) a partially ordered set $(A, \prec)$ of events ($\prec$ is a strict partial order):

$\forall a_1, a_2 \in A: a_1 \prec a_2$, if action a_1 occurs earlier than action a_2.

2. While there are false counterexamples:

2.1. Find action a that lead to the appearance of the counterexample. Find set A that contains action a: $a \in A$. In set

A find action b such that $b \prec a$.

2.2. Introduce a logical variable aux_b with the initial value $false$. In the model, replace b with the atomic sequence $b; aux_b \coloneqq true$.

3. By means of the logical AND, add aux_b to the guard of the command that contains action a. Replace a with the atomic sequence $a; aux_b \coloneqq false$.

For example, for one type of initial requests defined for the Elbrus-4C microprocessor, the set $(A, \prec)$ is as follows. Here, cc_i denotes the ith cache controller.

{ a_0 = processing of the previous request from process $cc_i, 1 \leq i \leq n$ is finished,
a_1 = requester cc_i sends an initial request,
$a_2 = system\_commutator$ receives the initial request,
$a_3 = system\_commutator$ sends snoop-requests to all $cc_j, 1 \leq j \leq n, j \neq i$,
$a_4 = cc_j$ receives a snoop-request, $1 \leq j \leq n, j \neq i$,
$a_5 = cc_j$ sends an answer to the snoop-request to the requester,
a_6 = the requester receives the coherent answer from cc_j,
a_7 = the requester sends the operation completion message to $system\_commutator$,
$a_8 = system\_commutator$ receives the operation completion message}.

The relation $\prec$ is defined as follows: $\forall i, j = 0, \dots, |A| - 1: i < j \Rightarrow a_i \prec a_j$. We identify the auxiliary variables with the elements of the set A.

Refinement of the abstract model of the Elbrus-4C cache coherence protocol required us to introduce two auxiliary variables, because there were two spurious counterexamples. Let us examine the introduction of the first variable.

The analysis of the first counterexample showed that the abstract process had sent the operation completion message to $system\_commutator$ before $system\_commutator$ received a coherent answer. Examination of the set A allows us to conclude that action a_7 happening at the wrong time led to the counterexample. According to the refinement procedure, in the set A we find action a_6 and introduce an auxiliary variable `ack_received` with the initial value $false$. Then we replace the operator that corresponds to a_6 with the atomic sequence consisting of this operator and the operator that assigns $true$ to `ack_received`. After this, we add `ack_received` to the guard of the command of the abstract process that contains a_7 and replace the operator that corresponds to a_7 with the atomic sequence consisting of this operator and the operator that assigns $false$ to `ack_received`. Thus, we guarantee that the behavior of the abstract process that led the false counterexample will no longer be exhibited.

C. Verification Technique

According to the results obtained by the author in this and the previous works, the proposed verification technique consists of the following steps (Figure 4):

1. Development of a concrete Promela model of the cache coherence protocol under verification. Using the proposed approach to model description, verification engineer develops Promela processes that model cache controllers and the system commutator and the necessary infrastructure elements (channel definitions, process creation). Specific actions performed by the processes correspond to the cache coherence protocol documentation.

2. Development of the abstract Promela model of the cache coherence protocol under verification. This step is performed automatically by the developed tool.

3. Verification of the abstract model. This step is the usual verification process of Promela models using the Spin model checker [17].

4. Analysis of the verification report generated by Spin. If there are no errors, then the verification process is finished with the conclusion that the cache coherence protocol is correct. If the report states the presence of an error, then the verification engineer should analyze the corresponding counterexample. If the engineer concludes that the counterexample is spurious because the corresponding sequence of steps is impossible in a real system, then the engineer refines the model in accordance with the proposed procedure and goes to step 3. Otherwise, if the counterexample represents an actual error in the cache coherence protocol, then the error is reported. When the protocol developers fix the error, the verification engineer incorporates the changes into the model and starts the verification process again (goes to step 1).

This sequence of steps is repeated until there are no counterexamples.

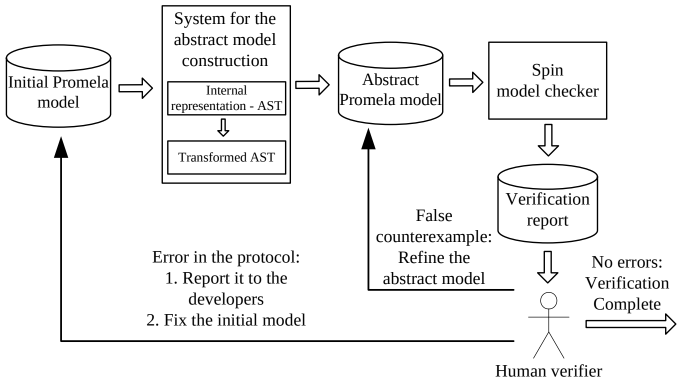

Initial Promela model
System for the abstract model construction
Internal representation - AST
Transformed AST
Abstract Promela model
Spin model checker
Verification report
False counterexample: Refine the abstract model
Error in the protocol:
1. Report it to the developers
2. Fix the initial model
No errors: Verification Complete
Human verifier

Figure 4 – Scheme of the Verification Process

V. Experimental Results

The proposed method was used to verify the MOSI family cache coherence protocol implemented in the Elbrus-4C computer system. The abstraction refinement step was completed after the introduction of two auxiliary variables.

Table 1 and Table 2 show resources consumed for checking the property

$$\mathbf{G}\{\neg(cache[1] = M \wedge cache[2] = M)\},$$

respectively, on the original and the refined abstract model. Spin's optimization COLLAPSE was used. The experiments were performed on an Intel Xeon E5-2697 machine with a clock rate of 2.6 GHz and 264 Gb of RAM.

Table 1. Required resources — original model

Number of cores	State space size	Memory consumption	Verification time
3	5.1×10^6	328 Mb	15 s
4	1.3×10^9	81 Gb	1.5 h

Table 2. Required resources — abstract model

Number of cores	State space size	Memory consumption	Verification time
any > 2	2.2×10^6	108 Mb	6.2 s

The tables show that even for $n = 3$ there is a gain in state space size and memory consumption. The needed amount of manual work is acceptable. Meanwhile, verification of the constructed abstract model means verification of the protocol for any $n \geq 3$. The task has been reduced to checking of $\sim 10^6$ states, which consumes ~ 100 Mb of memory.

VI. Conclusion

Many high-performance computers and most multicore microprocessors use shared memory and utilize complicated caching mechanisms. To ensure that multiple copies of data are kept up-to-date, cache coherence protocols are employed. Errors in the protocols and their implementations may cause serious consequences such as data corruption and system hanging. This explains the urgency of the corresponding verification methods.

The main problem when verifying cache coherence protocols (and other systems with a large number of components) by a fully automated method of model checking is state explosion. The article proposes a technique to overcome the problem for cache coherence protocols and make verification scalable. The price paid for scalability is acceptable, because the main ingredient – the verification method – is highly automated by the developed tool. Part of the method that requires manual work, namely, model refinement, can be done with a reasonable amount of effort, as shown by means of the Elbrus-4C protocol verification example. An approach to describing protocol models in Promela, a widely spread language in the verification community, is proposed. This approach lets us reflect the way protocol designers talk about protocols by representing protocols as a set of communicating finite-state machines.

The technique was successfully applied to the verification of the MOSI family cache coherence protocols implemented in the Elbrus-4C computer system.

Directions for future research include:

1. Development of methods and tools for verification of cache coherence protocols that are implemented by multiple levels of cache. The newest microprocessors (for example, Elbrus-8C, which employs the second- and third-level caches to implement cache coherence) define the need for such methods and tools.

2. Development of methods and tools for verification of hardware implementations of cache coherence protocols. In this direction, I have developed a tool that generates assembly code based on Promela models of cache coherence protocols. With this tool, I have found several dozen errors in the implementation of cache coherence in Elbrus microprocessors. Still, further research is needed to increase the level of confidence in design correctness.

References

[1] Patterson D.A., Hennessy J.L. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann, 2013. 800 p.

[2] Sorin D.J., Hill M.D., Wood D.A. A Primer on Memory Consistency and Cache Coherence. Morgan and Claypool, 2011. 195 p.

[3] Clarke E.M., Grumberg O., Peled D.A. Model Checking. MIT Press, 1999. 314 p.

[4] Burenkov V.S. Analiz primenimosti instrumenta Spin k verifikacii protokolov kogerentnosti pamyati (An analysis of the Spin model checker applicability to cache coherence protocols verification) // Voprosy radioehlektroniki. Ser. EVT. 2014. Vyp. 3. P. 126-134.

[5] Burenkov V.S., Kamkin A.S. Checking Parameterized PROMELA Models of Cache Coherence Protocols // Proc. of the Institute for System Programming. 2016. Vol. 28, Issue 4. P. 57-76.

[6] Burenkov V.S., Kamkin A.S. Metod masshtabiruemoi verifikacii PROMELA-modelei protocolov kogerentnosti kesh-pamyati (A Method for Scalable Verification of PROMELA Models of Cache Coherence Protocols) // Sb. trudov VII Vserossiiskoi nauchno-technicheskoi konferencii "Problemi razrabotki perspectivnih micro- i nanoelektronnih sistem". 2016. Chast II. P. 54-60.

[7] Burenkov V.S., Kamkin A.S. Applying Parameterized Model Checking to Real-Life Cache Coherence Protocols // Proc. of IEEE East-West Design & Test Symposium. 2016. P. 1-4.

[8] Burenkov V.S., Ivanov S.R. Metod postroeniya abstraktnih modelei, ispolzuemih dlya verifikacii protocolov kogerentnosti kesh-pamyati (A Method for Construction of Abstract Models Used for Verification of Cache Coherence Protocols) // Vestnik MGTU im N.E. Baumana. 2017. Vipusk 1. P. 49-66.

[9] McMillan K. Parameterized Verification of the FLASH Cache Coherence Protocol by Compositional Model Checking // Conference on Correct Hardware Design and Verification Methods, 2001. P. 179-195.

[10] Chou C.-T., Mannava P.K., Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols // Formal Methods in Computer-Aided Design, 2004. LNCS, Vol. 3312, P. 382-398.

[11] Krstic S. Parameterized System Verification with Guard Strengthening and Parameter Abstraction // International Workshop on Automated Verification of Infinite-State Systems, 2005.

[12] Talupur M., Tuttle M.R. Going with the Flow: Parameterized Verification Using Message Flows // Formal Methods in Computer-Aided Design, 2008. P. 1-8.

[13] O'Leary J., Talupur M., Tuttle M.R. Protocol Verification Using Flows: An Industrial Experience // Formal Methods in Computer-Aided Design, 2009. P. 172-179.

[14] Baier C., Katoen J.-P. Principles of Model Checking. The MIT Press. 2008. 984 p.

[15] de Guzman, J. Fastest numeric parsers in the world! http://boost-spirit.com/home/2014/09/03/fastest-numeric-parsers-in-the-world/.

[16] Spin Version 6 – Promela Grammar. http://spinroot.com/spin/Man/grammar.html.

[17] Holzmann, G. The Spin Model Checker: Primer and Reference Manual. Addison-Wesley. 2004. 608 p.

High Level Model-Based Test Generation for Digital Hardware

Mikhail Chupilko*, Alexander Kamkin*†‡, Mikhail Lebedev*, Sergey Smolov*
*Institute for System Programming of the Russian Academy of Sciences (ISP RAS)
25 Alexander Solzhenitsyn st., Moscow, 109004, Russian Federation
Email: {chupilko, kamkin, lebedev, smolov}@ispras.ru
†Lomonosov Moscow State University (MSU)
GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
‡Moscow Institute of Physics and Technology (MIPT)
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation

***Abstract*—The paper presents a new method for automated test generation for digital hardware descriptions. Test goals are stuck-at fault detection and a high level of design's source code coverage achievement. The method is based on model checking and data flow analysis. Extended finite state machines and high-level decision diagrams are used for modeling of hardware descriptions. Both categories of models are automatically extracted from the source code.**

I. Introduction

Functional verification and testing are resource- and time-consuming activities of the digital hardware design process [1]. To increase automation, *models of hardware designs* are frequently used. The models are mathematical abstractions that describe the target system's structure and behavior. There is a huge variety of verification-related problems that can be solved with the help of models: checking for the system's functional correctness [2], directed test generation [3], etc.

Today hardware design process consists of the following stages: 1) *architecture design*; 2) *detailed design*; 3) *logic synthesis*; 4) *physical synthesis*. The first stage includes requirement collecting and analysis. Requirements describe common structure of the system under development and formats of data exchanges between separate sub-components. At the second stage, the structure and cycle-accurate detailed description of the system behavior at the *register transfer level* (RTL) are described. This stage results in a digital hardware representation written in a hardware description language (HDL), like VHDL or Verilog [4]. The syntax of these languages is close to such traditional programming languages, like C or Ada. The third stage consists of a translation of the HDL description into a logic circuit format, for example, BLIF [5]. It represents a hardware design as a network of logical components called *gates* that are connected to each other. Each gate implements a Boolean function. The last stage is usually performed automatically by the modern CAD tools and results in a silicon chip layout. Due to the modern chips' increasing complexity, reduction of testing-related expenses (both in generation and in simulation) and early test generation tend to be challenging and most-wanted tasks.

The paper considers models that are automatically extracted from the source code of the target HDL design. Model-based tests show high level of source code coverage along with relatively short lengths in terms of simulation ticks [6]. But when applied during one of the abovementioned stages of the design process, like logic synthesis, they can decrease its effectiveness [7]. This results from the fact that high-level tests do not take into account the low-level *fault models*.

This paper continues the research that was initiated in [7]. Several test generation methods have been applied to detect so-called *mutants* - modifications of the target hardware description with injected errors. The error is considered to be detected by a test, when the mutant and the target descriptions have returned different output values for the same input test sequence. The method's error detection efficiency is calculated as the amount of mutants that have been detected.

The rest of the paper is organized as follows. Section 2 describes high-level models that are used in the proposed test generation method and gives a brief description of the fault model. Section 3 summarizes related works about the applying of model-based tests to low-level fault detection. Section 4 describes the proposed approach. Section 5 reports experimental results. Section 6 gives an idea of possible enhancement of the approach. Section 7 discusses the results of the work and concludes the paper.

II. Preliminaries

Let V be a finite state of *variables*. A *valuation* is a function that associates a variable $v \in V$ with a value $[v]$ from the corresponding domain. Let Dom_V be a set of all valuations of V. A *guard* is a Boolean function defined on valuations ($Dom_V \rightarrow \{true, false\}$). An *action* is a transformation of valuations: $Dom_V \rightarrow Dom_V$. A pair $\gamma \rightarrow \delta$, where γ is a guard and δ is an action, is called a *guarded action*. It is implied that there is a description of every function in some HDL-like language (thus, we can reason about not only semantics, but syntax).

An *extended finite state machine* (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 $\nu \in D_{VM}$ is referred to as a *context*, while a pair $(s, \nu) \in S_M \times D_{VM}$ is called a *configuration*. A transition t is said to be *enabled* for a configuration (s,ν), if $s_t = s$ and $\gamma_t(\nu) = true$.

Given a *clock* C (a periodic event generator) and an *initial configuration* (s_0,ν_0), the EFSM operates as follows. In the beginning, it resets the configuration: $(s, \nu) \rightarrow (s_0, \nu_0)$. On every tick of C it computes the set of enabled transitions: $T_E \leftarrow \{t \in T_M \mid s_t = s \cap \gamma_t(\nu) = true\}$. A single transition $t \in T_E$ (chosen nondeterministically) is enabled; the EFSM changes the configuration (updates the context and moves from the current state to the final one): $(s, \nu) \leftarrow (s'_t, \delta_t(\nu))$.

A *logic circuit* is a tuple $L = \langle G_L, V_L, E_L \rangle$, where G_L is a set of gates, V_L is a set of variables, and E_L is a set of *edges* between gates. Each gate $g \in G_L$ is a tuple (I_g, o_g, f_g), where $I_g \subseteq I_L$ is a set of gate inputs, $o_g \in O_L$ is a gate output, $f_g : Dom_{I_g} \rightarrow \{true, false\}$ is a Boolean function implemented by the gate g. Each edge $e \in E_L$ is a tuple (g_e, v_e, g'_e). where g_e and g'_e are respectively the *initial* and the *final* gates of e, whereas v_e is a variable which value is transmitted through the edge: $v_e \in (I_{g'_e} \cap \{o_{g_e}\})$.

For test generation method comparison a well-known fault model, called *stuck-at fault* model, is used. The idea is to corrupt Boolean functions of some gates in such a way, that they will return either $true$ (stuck-at-1) or $false$ (stuck-at-0) for all possible combinations of gate inputs. This model is frequently used for modeling manufacturing flaws at the physical synthesis stage.

III. Related Work

This chapter gives an overview of model-based test generation methods that are aimed at stuck-at fault coverage. [8] proposes an approach to functional test generation for VHDL designs. The method consists of the following stages: 1) automated translation of a HDL description into the *binary decision diagram* form; 2) stuck-at fault insertion into the BDD; 3) *distinguishing test* generation for a XOR composition of a fault-free BDD and a fault-containing BDD; 4) additional constraint generation for test application at the multi-module design level. For HDL-to-BDD translation, the approach uses the [9] method, that has some limitations in supported HDL coding style. In [10] a combined approach is proposed. It uses two kinds of models: a *high-level decision diagram* (HLDD) and an EFSM. An HLDD is a generalization of BDD; non-terminal nodes of the diagram could be marked not only by Boolean but arbitrary expressions. In the proposed approach an HLDD model is automatically extracted from an HDL description and a test is generated that covers all the diagram branches (for every HDL variable a separate diagram is extracted). Generated tests are passed to the next component, that simulates their execution on an EFSM model that is also extracted from the HDL code. For EFSM transitions that were not covered by the HLDD-based tests a *backjumping* technique is used to cover them too. The paper [6] proposes another EFSM-based approach that fixes several issues of [10] at test generation phase and uses another EFSM extraction approach. The experiments have shown that the new tests are shorter than the tests generated by [10], keeping the same HDL code coverage level. In [7] the method [6] has been experimentally evaluated along with another one, that uses ABC [11] equivalence checker for distinguishing sequences generation at BLIF level. The EFSM-based method demonstrates the higher HDL coverage level and shorter tests than ABC-based one, but lower levels of stuck-at fault coverage.

IV. Proposed approach

This paper is dedicated to the model-checking based approach [12] of the functional test generation for HDL descriptions and enhancements aimed at the increasing of stuck-at fault coverage level. The method flow is shown on the Figure 1.

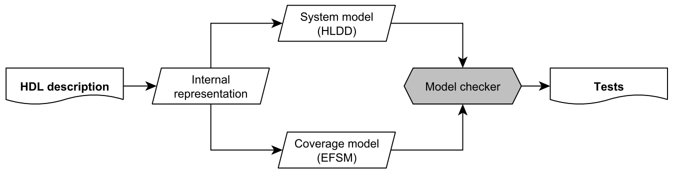


Fig. 1. Model checking-based method of test generation for HDL descriptions

The method uses two kinds of models that are extracted from the HDL code. The first one is a *system model* that is based on an HLDD formalism. The second one is a *coverage model* that is built from the EFSM. Both HLDD and EFSM models are automatically extracted from the HDL code. Some preliminary transformations are performed: the common *inner representation*, based on the control flow graph model, is built (see [12] and [13] for more details). The system model is a set of HLDDs related to every inner variable of the target HDL design. The coverage model is a set of conditions (*specifications*) extracted from the EFSM model. The EFSM model consists of the EFSMs that are extracted from every process of HDL description. The EFSM states contain Boolean expressions that are dependent on the *state-like registers* (SLR). The SLR are chosen with the help of dataflow-based heuristics, but can also be user-defined. The EFSM transitions are extracted from execution paths of the HDL description. All the EFSMs are *deterministic* – SLR-using expressions that are related to different states, are mutually incompatible.

EFSM-based specifications represent *reachability conditions* for EFSM transitions. The *linear temporal logic* (LTL) is used to describe them. Let $s \in S_M$ is a state of an EFSM M, $c(s)$ is a related SLR-using Boolean expression and $t \in T_M$ is an outgoing transition ($s_t = s$). In terms of LTL the reachability condition is as follows: $\neg\mathbf{F}(c(s) \wedge \gamma_t))$, where $\mathbf{F}(x)$ means that the x condition will become $true$ in a "future" moment.

Following the steps of the method [12], the system model and the coverage model are automatically translated into an input format of a model checker tool. The tool generates a number of *counterexamples* – sequences of input signals, that prove the contradiction between the specifications and the model. Since the specifications are formulated as negations, the counterexamples contain sequences of stimuli that reach the related EFSM transitions. Counterexamples can be easily translated into the HDL testbench form and can be executed by an HDL simulator.

The method is aimed at EFSM transition (and, again, HDL execution path) coverage. Another key feature of the method is flexibility in terms of specifications. Having another specifications, new kinds of errors can be covered by tests. So, the only thing to be changed to increase stuck-at fault coverage is the coverage model.

V. Experiments

The described model checking based method was implemented in Java programming language as a part of the Retrascope 0.2.1 software tool [15]. Fortress [16] formulae manipulation library and nuXmv [17] model checker were used. A subset of ITC'99 benchmark [18] was chosen for testing.

Three test generation methods have been compared: the described one (called "SMV"), the EFSM-based one (called "RETGA") [6], and the equivalence-checking based one (called "ABC") [7]. The last method generates distinguishing sequences for the mutants represented in BLIF format; the ABC [11] tool is used by this method.

Two metrics have been used for test comparison: the length in ticks and the fraction of detected mutants. A tool prototype called DTESK has been implemented for mutant generation. Taking the specified fault model type and the input HDL design, the tool generates a set of mutants along with simple *testbenches*. Every testbench contains one instance of the original design and one instance of the corresponding mutant. The testbench takes input values from the text file, passes them both to the mutant and the original design and then compares their outputs with XOR composition. If at some tick the comparison result is *true*, then the mutant is treated to be detected by the test.

Table I contains information about the ITC'99 designs that were used for test generation: the source code size, the corresponding formal model size (without specifications), the number of stuck-at fault mutants.

Table II contains the test-related information: the length in ticks and the percentage of mutants that were detected. Generated tests were simulated by the QuestaSim HDL simulator [19].

The experimental comparison results are as follows. For some designs (B02, B07) all the methods give 0% coverage of stuck-at fault errors. Such designs belong to the category of *untestable* [20] – their outputs are calculated in such a way, that the most part of the internal stuck-at faults cannot affect them. For several designs the described method reaches the

TABLE I
ITC'99 HDL Descriptions

Design	*HDL*	*SMV*	*Number of mutants*
B01	102	207	88
B02	70	143	48
B04	101	809	1342
B06	127	442	94
B07	92	370	784
B08	88	315	324
B09	100	263	284

same or close stuck-at fault coverage level (B01, B04, B06) as the leading one. Note, that in such cases model checking-based tests are frequently shorter than based on other methods. Finally, there are some designs (B08, B09), where the both HDL-based test generation methods reach lower levels of stuck-at fault coverage, than the ABC-based one. Additional efforts are needed to be done to cope with such effects. One possible improvement is described below.

VI. Future improvements

Since the proposed method uses the EFSM model, the stuck-at fault concept should be reformulated in these terms. According to the traditional methodology [14], let an EFSM model M of an HDL description contains a stuck-at fault, when for some transition $t \in T_M$ an assignment $a \in \delta_t$, that looks like $v := h(\bar{v})$, is substituted by $v := c$, where c is a constant. To make the difference between the error-containing and error-free EFSMs to be observable, the test should not only cover the corrupted transition, but transmit erroneous values to model outputs. So the EFSM transition reachability conditions should be substituted by *propagation trees* of erroneous values.

To give necessary formal definitions, data flow dependencies should be introduced. Let x and y be EFSM transitions ($x, y \in T_M$) and v is a variable ($v \in V_M$). The variable v is *defined* in x ($v \in Def_x$), when δ_x contains an assignment to v. The variable v is *used* in y ($v \in Use_y$), when v appears either in γ_y or on the right-hand side of some assignment(s) from δ_y. The transition y *depends* on the transition x ($y \in DEP(x)$), when $Def_x \cap Use_y \neq \emptyset$. Depending on the way the variable is used – in the guard or in the action – the *control* ($y \in DEP_c(x)$) and the *data* ($y \in DEP_d(x)$) dependencies are determined. Self dependencies are also acceptable.

A propagation tree P_t for the transition $t \in T_M$ is a directed acyclic graph $\langle T', D \rangle$. Graph nodes are EFSM model transitions from the set $T' \subseteq T_M$. Graph edges $d \in D$ are pairs of transitions (t_d, t'_d). The transition t'_d is a final node of the edge d, if and only if $Def_{t_d} \cap O_M \neq \emptyset$ or $t'_d \in DEP_d(t_d)$. The specified conditions mean: the tree transition either defines an output variable of the EFSM model, or is data-dependent on the parent transition. The propagation tree construction method is straightforward, for example, a breadth-first search can be used.

LTL logic can also be used to represent the propagation tree formalism in the proposed test generation method. The path from the tree root to one of the leaf nodes is as follows: $\phi =$

TABLE II
TEST GENERATION METHOD COMPARISON

Design	*ABC (ticks)*	*RETGA (ticks)*	*SMV (ticks)*	*ABC (%)*	*RETGA (%)*	*SMV (%)*
B01	227	49	37	90.91	98.86	90.9
B02	86	33	28	0	0	0
B04	–	36	56	–	99.93	99.93
B06	100	76	63	100	100	100
B07	133	166	162	0	0	0
B08	2745	52	31	98.77	79.94	44.44
B09	777	231	55	97.18	0	0

$\mathbf{F}((c(s_0) \wedge \gamma_0) \wedge \mathbf{F}((c(s_1) \wedge \gamma_1) \wedge ...\mathbf{F}(c(s_n) \wedge \gamma_n)...))$, where n is a number of path nodes. The propagation tree can contain more than one path, so the following specification should be used to check all the paths: $\neg\mathbf{F}(\phi_0 \wedge \phi_1 \wedge ... \wedge \phi_m)$, where m is a number of paths. Additional experiments are needed to estimate the effectiveness of this modification.

VII. CONCLUSION

The functional test generation method that is based on an automated extraction of formal models and checking them is described in this paper. The main advantage of this method is the testing purpose flexibility. Any other specifications can be formulated and checked to generate a test aimed at covering the corresponding property of a model.

The contribution of this paper is in applying the described method to stuck-at fault detection. The idea was to reuse tests generated for HDL designs in low-level hardware description, such as logic circuits (BLIF), and in testing.

The presented implementation of the method tends to produce shorter tests than existing approaches on the chosen hardware design set. Unfortunately, for some categories of designs, it shows small stuck-at fault coverage level. Additional improvements can be helpful, the one of them is described in the paper.

Future research directions include the method's application to more complex hardware designs. For such designs the problem of *test compaction* [21] becomes more and more important. The described approach uses a simple compaction technique, that is based on EFSM transition coverage estimation. An experimental comparison with existing methods in this field ([22]) should also be done.

VIII. ACKNOWLEDGMENT

Authors would like to thank Russian Foundation for Basic Research (RFBR). The reported study was supported by RFBR, the research project number is 15-07-03834.

REFERENCES

[1] J. Bergeron. *Writing Testbenches: Functional Verification of HDL Models*. Springer, 2003. 478 p.

[2] V.P. Ivannikov, A.S. Kamkin, A.S. Kossatchev, V.V. Kuliamin, A.K. Petrenko. *The Use of Contract Specifications for Representing Requirements and for Functional Testing of Hardware Models*. Programming and Computer Software, 33(5), 2007. 272-282 pp.

[3] P. Mishra, N. Dutt. *Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design*. Automation of Electronic Systems, 13(3), 2008. 1-36 pp.

[4] N.M. Botros. *HDL Programming Fundamentals: VHDL and Verilog*. Charles River Media, 2005. 506 p.

[5] *Berkeley logic interchange format (BLIF)*. Berkeley, U.C., Oct Tools Distribution 2, 1992, 197-247 pp.

[6] I. Melnichenko, A. Kamkin, S. Smolov. *An Extended Finite State Machine-Based Approach to Code Coverage-Directed Test Generation for Hardware Designs*. Proceedings of ISP RAS, 2015, 27(3), 161-182 pp.

[7] S. Smolov, J. Lopez, N. Kushik, N. Yevtushenko, M. Chupilko, A. Kamkin. *Testing Logic Circuits at Different Abstraction Levels: An Experimental Evaluation*. Proceedings of IEEE East-West Design Test Symposium (EWDTS), 2016. 189-192 pp.

[8] F. Ferrandi, F. Fummi, L. Gerli, D. Sciuto. *Symbolic Functional Vector Generation for VHDL Specifications*. Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 1999. 442-446 pp.

[9] S. Minato. *Generation of BDDs from Hardware Algorithm Descriptions*. Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1996. 644-649 pp.

[10] G.D Guglielmo, F. Fummi, M. Jenihhin, G. Pravadelli, J. Raik, R. Ubar. *On the Combined Use of HLDDs and EFSMs for Functional ATPG*. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2007. 503-508 pp.

[11] R. Brayton, A. Mishchenko. *ABC: An Academic Industrial-Strength Verification Tool*. Proceedings of the Computer Aided Verification Conference (CAV), 2010. 24-40 pp.

[12] A. Kamkin, M. Lebedev, S. Smolov. *An EFSM-Driven and Model Checking-Based Approach to Functional Test Generation for Hardware Designs*. Proceedings of IEEE East-West Design and Test Symposium (EWDTS), 2016. 60-63 pp.

[13] S. Smolov, A. Kamkin. *A Method of Extended Finite State Machines Construction From HDL Descriptions Based on Static Analysis of Source Code*. St. Petersburg State Polytechnical University Journal. Computer Science, Telecommunications, No.1(212), 2015. 60-73 pp.

[14] T. Chakraborty, S. Ghosh. *On Behavior Fault Modeling for Combinational Digital Designs*. Proceedings of International Test Conference, 1988. 593-600 pp.

[15] *Retrascope site* – http://forge.ispras.ru/projects/retrascope

[16] *Fortress site* – http://forge.ispras.ru/projects/fortress

[17] D. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, S. Tonetta, *The nuXmv symbolic model checker*. Proceedings of the 16th International Conference on Computer Aided Verification (CAV), No.8559, 2014, 334-342 pp.

[18] *ITC'99 benchmark site* – http://www.cad.polito.it/tools/itc99.html

[19] *QuestaSim site* – http://www.mentor.com/products/fv/questa

[20] X. Liu, M.S. Hsiao. *On Identifying Functionally Untestable Transition Faults*. Proceedings of the Ninth IEEE International High-Level Design Validation and Test Workshop, 2004, 121-126 pp.

[21] J.M. Vanfickell. *Stuck-at-fault Test Set Compaction*. Texas A&M University Undergraduate Research Fellows Thesis, 2004, 29 p.

[22] S. Eggersglüß, R. Wille, R. Drechsler. *Improved SAT-based ATPG: More Constraints, Better Compaction*. 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2013, 85-90 pp.

Verification of 10 Gigabit Ethernet controllers

Mikhail Petrochenkov[1], Ruslan Mushtakov[2], Irina Stotland[3]
Department of Verification and Modeling
MCST
Moscow, Russia
petroch_m@mcst.ru[1], mushtakov_r@mcst.ru[3], stotl_i@mcst.ru[3]

***Abstract* —This article proposes approaches used to verify 10 Gigabit Ethernet controllers developed by MCST. We present principles of the device operation and their characteristics. We describe a set of approaches used to verify such devices and provide the motivation for the chosen approach. The structure of the test systems that we used to verify devices and their components are presented and a set of test scenarios used to verify the device is described. Special importance in the process verification is the examination of network characteristics. Some approaches and techniques for throughput measuring are described. We provide list of found errors and their distribution by different types of functionality they affected.**

Keywords—10 Gigabit Ethernet, network interface controller, verification, throughput, UVM, test system.

I. Introduction

Development of modern computer networks provides the demand for high speed communication without sacrificing reliability. The evolution of Ethernet standard (IEEE 802.3 [1]) is an example of ever-rising demand for higher speed networks. Network interface controller (NIC) is the device that connects the computer to the network. Reliability and performance of the controller is very important for organization of modern networks. Network performance and accuracy of its work as a whole depends on the quality of implementation of NICs. To ensure that the controller satisfies all requirements for performance and reliability, it should be thoroughly verified.

Various methods of verification are used at all phases of NIC design flow. Common approaches for the device verification are physical prototype verification, system verification and stand-alone verification.

Process of **physical prototype verification** uses the device implemented in FPGA as a NIC in a "real" machine. Characteristics of the approach:

- Test stimuli are generated using operating system network drivers and signals from physical network (in our case - third-party 10 Gigabit Ethernet controllers).
- The fastest approach by a wide margin.
- Ability to execute "real life" scenarios and gather information to improve the device performance for most important use cases.
- Ability to debug network drivers.
- Difficulty in localization of detected errors.
- Slow iteration cycle due to slow recompilation to FPGA process.

In **the system verification** approach a NIC is simulated as a part of whole System on a Chip (SoC). NIC is configured according to required settings and then it executes network transactions. Characteristics of the system verification approach for the Ethernet controller:

- Test stimuli are memory access operation to the device registers and received Ethernet packets - very similar to typical mode of device operation.
- Simpler localization of detected errors and better error detection tools than using physical prototyping.
- Ability to implement directed scenarios from a physical prototype.
- Difficult and time consuming to cover all possible situations, especially for complex internal components.
- Requires all device components to be in working state.

In stand-alone verification a single device component is simulated, typically it is used for components with (1) high internal complexity and (2) which reliability is crucial for the device [2]. Properties of this approach:

- Test stimuli are transactions on the external interfaces of the component.
- Could be start as soon as only RTL-model of device component is ready (not the whole device).
- Faster simulation for smaller device subset.
- It is easier to create specific test cases and more complex test scenarios for component under test.
- Information about internal interfaces of the device is required.
- Can not eliminate the need of system verification completely and thus always requires extra labour and other resources.

In our previous projects we use only physical prototyping and system verification to verify NICs. However there are some types of errors that are hard to find using only these approaches. In this regard all aforementioned methods we used during verification process of 10 Gigabit Ethernet controllers. Physical prototype verification approach was used by separate team, and further discussion of it is beyond the scope of this article.

In this paper we present a case study for functional verification of 10 Gigabit Ethernet controllers developed by MCST. The paper addresses the problem and methods of stand-alone verification of 10 Gigabit Ethernet controllers.

The rest of the paper is organized as follows. In the Section 2 we describe the devices under test: RTL-models of the 10 Gigabit Ethernet controllers, their features and intended methods of implementation. Section 3 presents different test systems developed for components of the controller and the complete controller. In Section 4 we give further insight into the process of examination of the device network properties - most importantly its throughput. Section 5 presents the results of verification and plans of future work.

II. Device Under Test for Different 10 Gigabit Ethernet Controller implementation

Model of the 10 Gigabit Ethernet Controller is implemented using Verilog Hardware Description Language (HDL). It is RTL (register transfer level) description that is used in different implementation (Device Under Test, DUT):

- FPGA-based network controller (based on Altera Cyclone V [3]). This FPGA provides a set of components that were used in the device: PCI Express Hardware IP module that implements physical and data link layer of the protocol, and a set of configuration space registers, and XAUI Hardware IP module to transform 10 Gigabit Media Independent Interface (XGMII) signals. It is connected to the other parts of the system using standard interface Avalon [4].
- ASIC-based network controller - a part of a currently developed Elbrus-16C System on Chip (SOC). Controller is connected to the rest of the system using in-house interface (SLink) to transfer packets based on PCI Express transaction layer packets.

General schemes of both DUT are presented in figure 1. Both types of the DUT share the Ethernet Control Module and implement same programming interface. This interface is typical for PCI and PCI-Express devices. A set of memory-mapped registers are used to control device behavior. Those registers can be separated into four groups:

1) PCI Express registers - common set of registers of PCI Express devices. They are used to control access to internal memory of the devices and from the device to system memory. It also implements basic interrupt control.

2) Media Access Control (MAC) registers allow to control the Ethernet physical layer. They are used to control pause frames, control sum (CRC) calculation, non standard-compliant frames and limit the speed of packet transmission.

3) Transceiver (TX) registers control the transmission of packets from the system to the network, calculation of CRCs for higher level protocols supported by the controller (IPv4 and IPv6, TCP and UDP).

4) Receiver (RX) registers control the reception of packets from the network, packet filtering and control sum checking for IP, TCP/IP and UDP/IP packets.

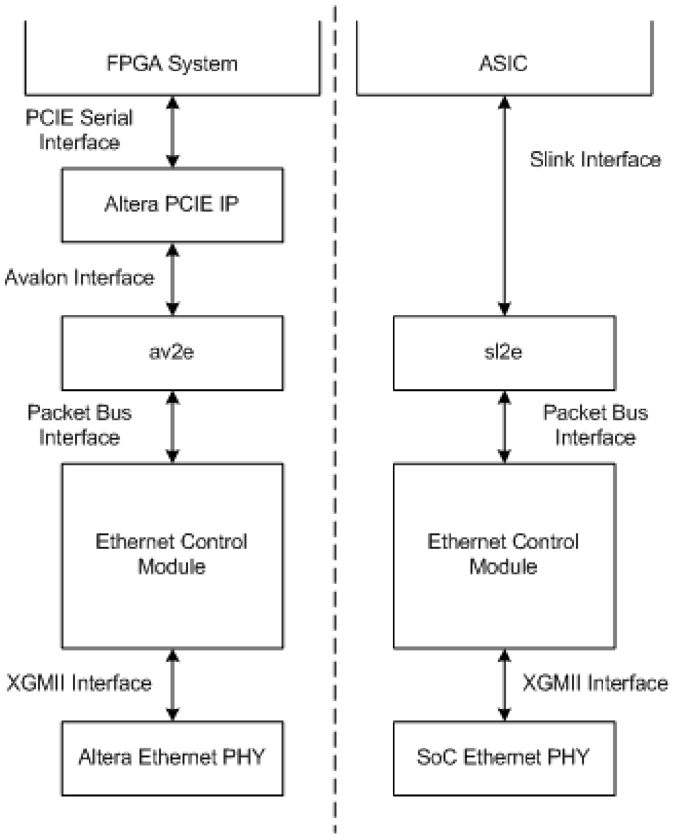


Fig. 1. Devices under test for FPGA and ASIC-based in SoC implementation.

To allow multiple processes to work in parallel with a single controller TX and RX registers contain several identical groups of registers (descriptor queues).

Ethernet Control module uses universal packet bus interface. This interface provides convenient access to large continuous areas of memory where Ethernet packet data are stored. Altera PCI Express and SLink interfaces work with packets of size up to 64 bytes. To increase the rate of data transmission and reduce the CPU involvement into device operation it used direct memory access (DMA). Devices use different modules to connect packet bus to Avalon and SLink interfaces. Internal names of those connector modules are av2e (avalon to everything) and sl2e (SLink to everything). Those connectors implement DMA by splitting packet bus transactions and transforming them into memory access operations.

III. Test systems for 10 Gigabit Ethernet controller verification

As stand-alone verification could be started as soon as RTL-model of device component is ready, without waiting RTL-model of the whole device. So verification of the 10 Gigabit

Ethernet Controller process was started at the same time as the development of the FPGA-based controller and SoC. This approach allowed identifying errors earlier, and reducing total development time of the device. To check correctness of the CC it is included in a test system — a program that generates test stimuli, checks validity of reactions and determines verification quality.

There are several verification methodologies in order to develop constraint-random coverage-driven verification test systems. A verification methodology provides guidelines, class libraries and macros libraries. The Universal Verification Methodology (UVM) [5] is currently the most widespread verification methodology. UVM allows automating test system design process and makes it easier to add new components and collecting the functional coverage [6]. In paper [7] the approach to UVM test system developing for Gigabit Ethernet is presented. However Gigabit Ethernet has some differences in protocol and interfaces from 10 Gigabit Ethernet. Moreover in our case we have to verify in-house Slink interface communication.

For stand-alone verification of the 10 Gigabit Ethernet Controller we developed two stand-alone UVM test systems based on two different DUTs for FPGA and ASIC-based implementation. The structures of the test systems and approaches used to process verification both of DUTs are presented below.

A. Test Systems for FPGA-based 10 Gigibit Ethernet Controller Verification

The top level module of the 10 Gigabit Ethernet Controller is called XGBE (10 GigaBit Ethernet). The structure of the test system for XGBE stand-alone verification is provided in figure 2.

In the controller, packet is represented as one or multiple (split) descriptors and a payload stored in the system memory. Each transmit and receive descriptor queue in the device works with continuous area of memory where descriptors are stored. The controller implements head and tail pointer registers used to determine descriptors currently in use. Each descriptor contains a payload pointer. XGBE driver transforms Ethernet packet between representation used by the controller and representation in the test system - UVM-base transaction class.

Algorithm for packet transmission:

1) Allocate memory for payload.

2) Form corresponding packet descriptor.

3) Write descriptor(s) to first free memory location in a queue.

4) Change queue head pointer value.

5) Wait until tail value becomes equal to head.

6) Collect packet transmission information and free used memory resources.

Packet reception algorithm works in a similar way, but because we do not have an information on expected packets sizes, algorithm works in two threads: descriptor preparation and packet reception. Descriptor preparation works as follows:

1) Wait when test system requests additional space for packet reception (conditions of this request are generally test-specific and determined by test system settings).

2) Allocate memory for the number of descriptors and fill corresponding descriptor memory.

3) Increase RX queue head value.

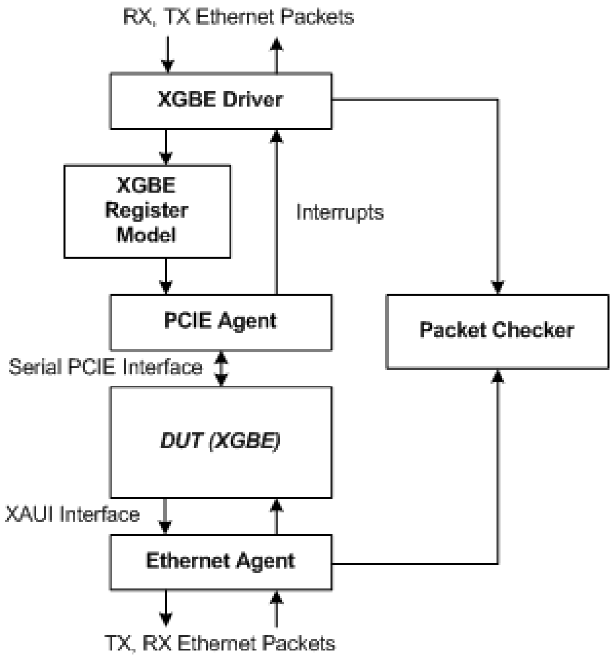


Fig 2. Structure of XGBE test system.

Packet reception algorithm:

1) Wait for change of RX queue tail value.

2) Collect received packet data from descriptor and payload.

3) Free resources used allocated in descriptor preparation routine.

To simplify access to device registers and abstract away details of register access operations, UVM register model (XGBE Register Model) of the controller was developed. This model uses a bus adapter to transform generic register access operations to the required bus format (in our case - transactions for PCIE agent). Other features of PCIE agent used by device driver are direct access to system memory and interrupt notifications.

Test system was used to verify the device on various test sequences, directed at different device functions. Test can be separated into four large groups: data flow tests, filtering tests, packet parsing tests and throughput tests. The general goal of the first group is to ensure that data processed by the controller will stay correct. At first, maximum possible packet flow through the device was tested. Later we started introducing different bottlenecks (by means of Ethernet Pause frames, PCI Express credits, limiting the size of transmission and reception buffers, available amount of receive descriptors etc...) to achieve different events in the internal components of the controller. The goal of the second group is to ensure correctness of filtering capabilities of device. A set of packets are generated in a way to be test all available packet filters. For the third group, higher-level protocol packets are encapsulated in the basic Ethernet frame and the ability of the device to handle them correctly (packet type detection, automatic checksum calculation etc.) was tested. Throughput tests will be discussed separately later in the article.

It also was decided that the stand-alone verification was necessary for a single type of module in the device - connector between multiple internal packet buses and the external PCI Express like interface (Altera Avalon Interface of PCI Express module), due to several reasons:

- These modules are relatively independent from the rest of the system, its early completion allowed for early verification start.
- High complexity of this module is due to complex rules of transaction splitting.
- Different interactions between all packet bus requesters are difficult to achieve in a complete system.

Its structure is provided in a figure 3. Connector module communicates through Altera PCIE module with the memory inside the PCI Express agent. Information about Ethernet packets is stored in this memory. These data can be separated into two groups: packet descriptors (which are used by the controller to facilitate data transfer) and payload of packets themselves. Connector module solves several problems. First, it transforms requests from packet bus to PCIE memory access transactions. Second, it transforms the responses to those requests, to format convenient for the rest of the devices. It was decided to include third party PCIE module that (we presume) is well-tested and bug-free because we have access to PCI Express agent, but do not have one for the Avalon interface. Additional performance gained through exclusion of this module is compensated by time and other resources needed for development of Avalon agent.

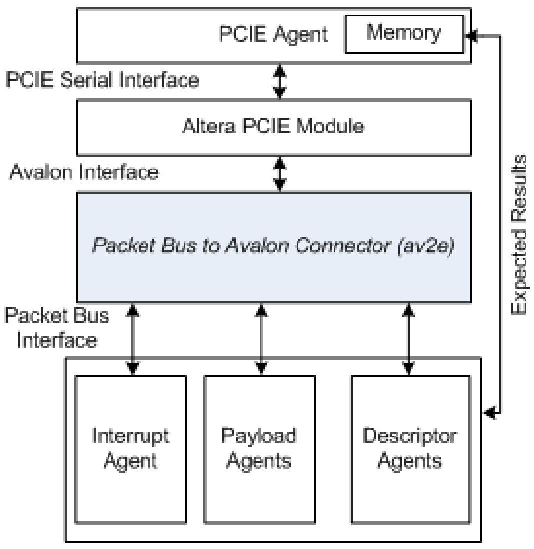


Fig 3. Structure of av2e module test system.

To verify av2e module (connector to Avalon interface) as a part of 10 Gigabit Ethernet controller the test system was designed. The test system is also designed using UVM and consists of the set of components, which are inherited from standard classes of UVM library.

The exchange between these components is carried out by transactions that facilitates scaling and configurability of system. Developed components could be adapted for use in system for verification of a whole controller to speed up its development.

B. *Test System for ASIC-based 10 Gigibit Ethernet Controller Verification*

Test system for connector between packet buses and system SLink interface module (sl2e) is similar to one used for verification of av2e module. PCI Express agent and (and Altera PCI Express module) were replaced with SLink agent which provides similar interface for other parts of test system:

- Transformation of transaction-level PCI Express operations into interface signals.
- Access to internal memory of the agent.
- Automatic generation of the completions for upstream requests.
- Notification mechanism for special requests.

The test system for ASIC-based 10 Gigabit Ethernet controller was developed by replacing PCIE Agent with SLink agent.

Usage of various test system components by different test systems is summarized in table 1. Only limited part of device register model (PCI Express Configuration space registers) was actively used in av2e and sl2e test systems.

TABLE I. TEST SYSTEM COMPONENTS REUSE

	FPGA XGBE	av2e	ASIC XBBE	sl2e
Bus Agent	-	+	-	+
PCIE Agent	+	+	-	-
Slink Agent	-	-	+	+
Register Model	+	+/-	+	+/-
XGBE Driver	+	-	+	-

IV. THROUGHPUT ANALYZING

Throughput is one of the most important characteristics of any network controller. In our case the controller should support throughput of 10 Gigabit per second for packet transmission and reception. Therefore, tests system must support the development of special scenarios for throughput testing. It is implemented in the test system by limiting the test system and device configuration in a way that it will not introduce new “bottleneck”.

To achieve necessary controller throughput it is essential to ensure that every component satisfy the requirement. In the controller PCI Express Gen2 x4 bus was used as a connection to the system. Maximum possible value of throughput for this bus is 16 Gigabit per second. Thus, this bus satisfies the requirements. Throughput of av2e module was measured after its verification was complete. To do that, the throughput analyzer was developed. It executes all necessary calculations using the information about the start time of first packet`s data transmission and the end time of the last one. Initial value of av2e module was ~11.2 Git/s in both directions. This value is higher than maximum packet flow from the network, so this module will not serve as a “bottleneck”.

Measurements of throughput of whole controller started after the verification of single packet transfer. Separate packet analyzers were designed for transmission to network and reception from it. Principle of measurement is similar to the one described above: analyzer collects the information on transmission start time, end time and size of transferred data. A set of tests were developed to determine throughput for different packet flows: (1) transmission, (2) reception, (3) mixed and (4) loopback. In addition, for each of these tests it is possible to select mode of operation: either raw Ethernet packets of mixed UDP/TCP flow to ensure that packet parsing will not slow down the device.

The first measured value of throughput was 0.36 Gbit/s in loopback mode. This, of course, does not satisfy the requirements. After multiple performance improvements the goal to achieve maximum possible throughput for separate transmission and reception packet flow was achieved. At this moment, throughput value for mixed mode 10Gbit/s for reception and 4Gbit/s for transmission and 7 GBit/s for both values in loopback mode. Work to further improve the performance is ongoing.

V. Results

To verify the 10 Gigabit Ethernet Controllers four separate test systems were designed using a set of components. Test sequences were developed to test the correctness of the device for each test system.

A total of 49 test scenarios were used to verify all functions of av2e module: read and write operations with different parameters, sequentially and in parallel. Total number of bugs detected by the av2e test system is 14. Found errors are the corruption of transmitted or received data and complete loss of packets by the components.

Number of test for the whole 10 Gigabit Ethernet controller is 31.They thoroughly check that the device works as described in the specification. Different test scenarios check different modes of operation: transmission of packets from system to network, reception of packets from network, mixed flow and loopback mode. The check proper handling of different types of payload (Raw Ethernet, or encapsulated IPv4, IPv6, UDP, TCP, Runt Frames, PTP packets), working with packets with different priorities, working with packets with vlan tags, pause frames, checking of packet filtering capabilities and automatic calculation of checksums. Different ways of interaction with the system are also checked: correctness of interrupts and mirroring of certain device registers in memory.

As a result of verification of the controller 74 errors were discovered and corrected. Those can be divided into 3 groups:

- Errors in data transmission is the biggest group of 49 errors. Those errors caused the transmission of incorrect data in packets by the controller. This group includes such errors as: partial loss of data, duplication of received data, “merging” of different packets into one and incorrect calculation of checksum.
- Number of errors in packets parsing and filtering is 9. They caused incorrect detection of types of packets, incorrect placement of CRCs in packet and incorrect filtering of received packets.
- Group of 16 miscellaneous errors have not caused errors in packet transfer. Those errors appeared during accessing internal registers of the device or inter and caused suboptimal utilization of the device resources.

All above-mentioned errors were corrected. ASIC-based version of the device and its test system are currently under active development. Our future works is aimed at further verification of ASIC-based version of the 10 Gigabit Ethernet Controller, developing UVM-based reusable components (UVC) for PCIE, Avalon, Slink interfaces for using in test system for other network controllers.

References

[1] IEEE Standard for Ethernet. IEEE Std 802.3-2012. 1983 p.

[2] Petrochenkov M., Stotland I., Mushtakov R. Approaches to Stand-alone Verification of Multicore Microprocessor Caches. Trudy ISP RAN, vol. 28, 3, pp. 161-172.

[3] Cyclon V – Overview. URL: https://www.altera.com/products/fpga/cyclone-series/cyclone-v/overview.html (09.04.2017).

[4] Avalon Interface Specification. Altera. MNL-AVABUSREF. 2015.12.10. 101 Innovation Drive. San Jose, CA 95134. URL: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/mnl_avalon_spec.pdf (09.04.2017).

[5] Standard Universal Verification Methodology. URL: http://accellera.org/downloads/standards/uvm (09.04.2017).

[6] Stotland I., Shpagilev D., Petrochenkov M. Osobennosti funkcional'noj verifikacii kontrollerov vysokoskorostnyh kanalov obmena mikroprocessornyh sistem semejstva "Elbrus" [Features of High Speed Communication Controllers Standalone Verification of “Elbrus” Microprocessor Systems]. Voprosy radioelektroniki, seriya EVT, 2017, 3, pp. 69-75.

[7] S. Chitti, P. Chandrasekhar, M. Asha Rani. “Gigabit Ethernet Verification using Efficient Verification Methodology”. Proc. of International Conference on Industrial Instruments and Control (ICIC), College of Enginnering Pune, India. May 28-30, 2015, pp.1231-1235.

Creating Test Data for Market Surveillance Systems with Embedded Machine Learning Algorithms

Olga Moskaleva
Exactpro, LSEG, Russia,
Email: olga.moskaleva@exactprosystems.com
http://www.exactprosystems.com/

Anna Gromova
Exactpro, LSEG, Russia,
Email: anna.gromova@exactprosystems.com
http://www.exactprosystems.com/

Abstract—**Market surveillance systems, used for monitoring and analysis of all transactions in the financial market, have gained importance since the latest financial crisis. Such systems are designed to detect market abuse behavior and prevent it. The latest approach to the development of such systems is to use machine learning methods. The approach presents a challenge from the standpoint of quality assurance and the standard testing methods. We propose several types of test cases which are based on the equivalence classes methodology. The division into equivalence classes is performed after the analysis of the real data used by real surveillance systems. This paper describes our findings from using this method to test a market surveillance system that is based on machine learning techniques.**

Keywords—test data, equivalence classes, market surveillance systems, machine learning.

I. INTRODUCTION

A. *Market surveillance systems*

Electronic trading platforms have become an increasingly important part of the financial market in recent years. They are obligated to take legal responsibilities [1], [2] and correspond to the law and the regulatory requirements. Therefore, all market events in the contemporary electronic trading platforms are monitored and analysed by market surveillance systems.

Such systems are designed to detect market abuse behavior and prevent it. Their main goals are detection and prevention of such market abuse cases as insider trading, intentional and aggressive price positioning, creation of fictitious liquidity, money laundering, marking the close, etc. [3]. Different data mining methods are used for improving the quality of the surveillance systems work [4], [5], [6], [7], [8], [9].

B. *Quality assurance for market surveillance systems*

The standard quality assurance (QA) methods and technologies seem to be powerless in regard to machine learning (ML) applications. C. Murphy, G. Kaiser, M. Arias even introduced a concept of "non-testable" applications [10]. From the QA perspective, we do not have to test whether an ML algorithm is learning well, but to ensure that the application uses the algorithm correctly, implements the specification and meets the users expectations. In this paper, we employ the term testing in accordance with the QA theory.

It is clear that a sufficient input data set is needed for high-quality testing coverage. Furthermore, the testing data set should be as close as possible to the real data or should even be real. So, which approach should be used for creating data to verify the implementation of an ML-algorithm more fully?

We can test a market surveillance system in the following ways:

1) By creating test cases which are based on the knowledge of the business rules from the specification. Such test data are similar to the real users behaviour.
2) By generating various datasets which contain different combinations of variables.

Both variants are suitable for the surveillance systems that use standard control flows, like loops or choices. For standard systems, there is a set of rules, which allows getting a clear output result for specific input data. When it comes to intelligence systems, it is not normally obvious what will happen as a result of certain input because an ML algorithm builds its own dependencies between the variables and human interpretation of such dependencies is impossible. Because of this, it is important to be able to create a set of test cases that will generate obvious and predictable output. Therefore, the second approach to generating the test data allows for the creation of output that is easily interpretable.

C. *Contribution*

This paper introduces the following contribution:

1) Creating a model for lassifying the transactions. This model can be used for the detection of market manipulations.
2) Test cases generation for defining the weaknesses of the created model. The test cases are based on the equivalence classes.
3) Testing the prototype based on the created model and the analysis of the received results.

II. RELATED WORK

A. *Ongoing problems in the quality assurance of the modern surveillance systems*

It is known that the system containing ML algorithms should learn using a dataset that is real or close to real. Obviously, for testing purposes, it is necessary to use a dataset with a similar structure.

Moreover, during the creation of this dataset, it will be helpful to emphasize the variability of values of the attributes included in the sampling. By generating a variety of combinations with different values, we will create test cases to find out the weaknesses in the ML algorithm, which are related to the separation of classes.

B. Existing approaches

C. Murphy, G. Kaiser, M. Arias proposed to divide the data into equivalence classes to generate test cases for testing the ”non-testable” software [10]. It should be noted that forming equivalence classes is a standard approach in quality assurance [11].

However, C. Murphy, G. Kaiser, M. Arias suggest to follow three steps in testing this kind of software. Firstly, the data should be divided into several classes, taking into account the size of the dataset, the potential ranges of the attributes and the label values, etc. Then, the test cases, that are based on the investigation of the ML algorithm used in this system, should be created. At last, several testing datasets should be generated.

Supposing that a dataset for QA purposes has been defined, based on the knowledge of the method used in the machine learning system. For solving the problem of the dataset sufficiency, C. Murphy, G. Kaiser, M. Arias propose a parameterized random data generation methodology. This methodology enables us to generate large datasets and randomly control them [12], [13], [14].

In their paper, J. Zhang et al. suggest to use predictive mutation testing, as it allows to make decisions without executing the costly mutation testing [15].

Thus, it becomes clear that new methodologies for testing ML applications are being developed. However, the contemporary market surveillance systems impose additional requirements on them. These systems should be able to self-detect the incorrect behavior and classify alerts, and further point at the initiator of this behavior. This should always be noted during the process of creation of test scenarios.

III. Definitions and assumptions

A. Structure of the transaction log

A transaction is an event that happens on the financial market and changes any financial instrument, for example:

- submitting a buy-order for security S with price P and volume V;
- cancelling an order with id I;
- trading on security S at price P with volume V, etc.

Transaction logs are the data that are used by the ML surveillance system. Each transaction is stored as an object and has a set of input parameters (transaction characteristics) and one output parameter (presence or absence of the alert):

$I = \{i_1, i_2, ..., i_j, ..., i_n\}$, where $i = \overline{1, n}$;

$i_j = \{TID, B, IID, Side, CP, ExP, ExS, TV, S, TInt, Alert\}$.

Where:

$TID \in N$,

$B = \{Broker_1, Broker_2, ..., Broker_k\}$, where k is the number of brokers that are available in the configuration file,

$IID = \{Instrument_1, Instrument_2, ..., Instrument_m\}$, where m is the number of instruments that are available in the configuration file,

$Side = \{Buy, Sell\}$,

$CP \in Q$,

$ExP \in Q$ and $ExP \geq 0$,

$ExS \in Z$,

$TV \in Z$,

$S = \{OAC, RT, CAC, ReOAC, ResumeAC, Halt\}$,

$TInt \in N$,

$Alert = \{0, 1\}^M$.

For more attribute details, please refer to Table 1.

TABLE I.

Parameter	Type	Comment
Transaction ID	numerical	Unique identifier of transaction
Broker	categorical	Company to which the trader belongs
Instrument ID	categorical	Identifier of the instrument to be traded
Side	categorical	Side of the order
Change Price	numerical	ChP=LastTradedPrice-CurrentPrice
Executed Price	numerical	Price of the trade
Executed Size	numerical	Volume of the Trade
Total Volume	numerical	Total volume traded on the instrument
Session	categorical	Current session on the instrument
Time Interval	interval	Time interval between transactions
Alert	boolean	The output: 0 - regular transaction,1 - suspicious transaction

Each transaction can cause several different alerts, that is why every alert should be classified. This type of classification is called a classification with overlapping classes.

B. Market abuse alert

The following type of behavior can be considered as abusive: a broker tries to raise or bring down the price on several trades. The alert will be triggered if the price deviation and the traded volume reach the threshold values. If the price goes up, the buy-orders should be checked, if the price goes down – the sell ones.

IV. Background

As each transaction is stored as an object and has a set of independent variables and one dependent variable, for testing purposes, these data should be divided into several equivalence classes. All the objects in one equivalence class have the same characteristics of certain attributes [11] and trigger the same system behavior.

It is important to analyze the transactions, the system behavior and the meaningful parameters before forming the classes. We can use the following types of test cases which are based on equivalence classes. These classes are defined after the analysis of real data used by the surveillance systems:

1) **Consistency checks** with consideration of categorical transaction attributes ($Firm$, $Session$, $Action$, etc.):

a) Let $a = a_1$ and $class = 1$, and $a = a_2$ and $class = 0$. The categorical attribute gets a concrete value for several test cases, but the other transaction attributes have different values. For such combinations, the class value is set to 1. This is a check for how strongly the $class$ value correlates with the concrete value of the I-th attribute.

b) Let a categorical attribute get any values, always leaving the $class$ set to 1. Let attribute $b = b_1$ OR $b = b_2$ OR $b = b_3$, $class = 1$. It checks that the system can assume that this attribute is inessential.

c) Let $0 \leq c \leq 10$, $class = 1$. The same as for (b), but for numeric variables.

d) Let $c \geq 0$, $class = 1$. The same as for (b), but for numeric variables.

2) **Checks for empty values.** Due to the fact that a surveillance system analyses all the transactions on the financial market and that the considered transaction can be different for each type of alert, it is possible that some values will be empty.

3) **Checks for the presence of noise.** We perform checks for the presence of noise using the equivalence classes for numeric transaction attributes. Some attributes can have values in the concrete range, but mostly they take the average value. For example, the price percentage variable takes the values from 0 to 100 with the mean of 50 and has a low variance. Even if the border values (0 and 100) are included in the acceptable range, they appear so rarely that we treat them as frequency emissions.

V. Approach

A. Classification model of market abuse

To prove that the proposed methodology is effective, we have created a model which allows to classify the transactions by one type of abusive behavior. It should be noted that we have to deal with rather specific data, i.e. financial transactions.

It is important to make a thorough analysis of the data related to the business logic, the system requirements and the structure of transaction logs, to be able to define the attributes and their particular qualities. The correct outcomes can only be provided when considering all these factors. Thus, this particular dataset structure, as it is presented in Table 1, was selected for this type of abusive behavior.

We extracted 639 transactions from the messages of one of the electronic trading protocols and manually classified them, using these data to build the classifier. We used a decision tree algorithm for our research. After that, the classifier was trained on a set and its performance was evaluated.

Precision, recall, and F-measure are the evaluation metrics that we used for our calculations. Values of these metrics that we received are represented in Table 2.

TABLE II.

Precision	Recall	F-measure
0.615	0.678	0.645

B. Prototype Testing

To test our prototype, we have created a dataset with the same structure as the training dataset and performed all the validations described in Section 4. For each type of the validation, we created the test cases based on the equivalence classes. Please refer to Table 3 for detailed examples.

Figure 1 illustrates the proposed approach. Test Data is the whole set of data that will be used to test ML-based surveillance system. Then these data are divided into equivalence classes, as proposed in Section 4. The generated test cases are used for testing Market Surveillance Systems based on ML.

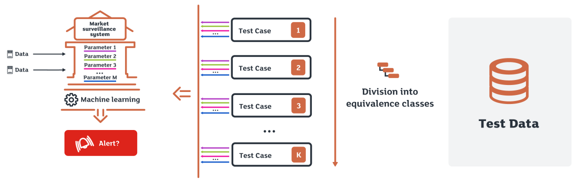


Fig. 1. Prototype Testing

VI. Results

After testing our prototype, we received the values of the evaluation metrics that are presented in Table 4.

After a detailed analysis of the instances where the model detected an error, it was observed that errors occurred in:

- different Firms - Scenario 2: Consistency checks.
- empty values - Scenario 5: Checks for empty values.
- border values - Scenario 6: Checks for the presence of noise.

Thus, the test that we performed revealed some weaknesses in the ML algorithm:

1) An error occurred in one of the categorical attributes during the consistency checks. It may indicate that the model does not take into account the categorical attribute in the algorithm.
2) Classification errors occurred in empty values, so the algorithm may miss some abnormal behavior when encountering them. For example, all manipulations at the start of the day may be missed by the surveillance system.
3) Some errors were detected in the border values used for the Executed Size variable. According to the general quality assurance theory, we should validate the border values for numeric variables. But in our experiment, unlike what was expected, we saw that the algorithm failed to validate such test cases. The reason is possibly linked with the distribution of the training sample.
4) Errors in the tests took place due to thoughtless randomization. After analyzing the results, we can conclude that it is crucial to randomise the dataset with the business logic in mind.

TABLE III.

Scenario 1: Consistency checks 1.a. Instrument		
Test case 1	Test case 2	Test case 3
$B = Broker_1$	$B = Broker_2$	$B = Broker_3$
$IID = Instrument_1$	$IID = Instrument_1$	$IID = Instrument_1$
$Side = Buy$	$Side = Sell$	$Side = Sell$
$CP = CP_1$	$CP = CP_1 - 1$	$CP = CP + 2$
$ExP = P_1$	$ExP = P_2$	$ExP = P_3$
$ExS = S_1$	$ExS = S_2$	$ExS = S_3$
$TV = TV_1$	$TV = TV_2$	$TV = TV_3$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_2$	$TInt = TI_3$
$Alert = 1$	$Alert = 1$	$Alert = 1$
Scenario 2: Consistency checks 1.b Firm		
Test case 1	Test case 2	Test case 3
$B = Broker_2$	$B = Broker_3$	$B = Broker_1$
$IID = Instrument_1$	$IID = Instrument_1$	$IID = Instrument_1$
$Side = Buy$	$Side = Buy$	$Side = Buy$
$CP = CP_1$	$CP = CP_1$	$CP = CP_1$
$ExP = P_1$	$ExP = P_1$	$ExP = P_1$
$ExS = S_1$	$ExS = S_1$	$ExS = S_1$
$TV = TV_1$	$TV = TV_1$	$TV = TV_1$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_1$	$TInt = TI_1$
$Alert = 1$	$Alert = 1$	$Alert = 1$
Scenario 3: Consistency checks 1.c ExPrice		
Test case 1	Test case 2	Test case 3
$B = Broker_2$	$B = Broker_2$	$B = Broker_2$
$IID = Instrument_1$	$IID = Instrument_1$	$IID = Instrument_1$
$Side = Buy$	$Side = Buy$	$Side = Buy$
$CP = CP_1$	$CP = CP_1$	$CP = CP_1$
$ExP = P_1$	$ExP = P_2$	$ExP = P_3$
$ExS = S_1$	$ExS = S_1$	$ExS = S_1$
$TV = TV_1$	$TV = TV_1$	$TV = TV_1$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_1$	$TInt = TI_1$
$Alert = 1$	$Alert = 1$	$Alert = 1$
Scenario 4: Consistency checks 1.d. TV		
Test case 1	Test case 2	Test case 3
$B = Broker_2$	$B = Broker_2$	$B = Broker_2$
$IID = Instrument_1$	$IID = Instrument_1$	$IID = Instrument_1$
$Side = Buy$	$Side = Buy$	$Side = Buy$
$CP = CP_1$	$CP = CP_1$	$CP = CP_1$
$ExP = P_1$	$ExP = P_1$	$ExP = P_1$
$ExS = S_1$	$ExS = S_1$	$ExS = S_1$
$TV = TV_1$	$TV = TV_2$	$TV = TV_3$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_1$	$TInt = TI_1$
$Alert = 1$	$Alert = 1$	$Alert = 1$
Scenario 5: Checks for empty values. ExP		
Test case 1	Test case 2	Test case 3
$B = Broker_2$	$B = Broker_3$	$B = Broker_1$
$IID = Instrument_1$	$IID = Instrument_2$	$IID = Instrument_3$
$Side = Buy$	$Side = Sell$	$Side = Buy$
$CP = CP_1$	$CP = CP_1$	$CP = CP_1$
$ExP = empty$	$ExP = empty$	$ExP = empty$
$ExS = S_1$	$ExS = S_2$	$ExS = S_3$
$TV = TV_1$	$TV = TV_2$	$TV = TV_4$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_1$	$TInt = TI_1$
$Alert = 1$	$Alert = 1$	$Alert = 1$
Scenario 6: Checks for the presence of noise. ExS.		
Test case 1	Test case 2	Test case 3
$B = Broker_2$	$B = Broker_2$	$B = Broker_2$
$IID = Instrument_1$	$IID = Instrument_1$	$IID = Instrument_1$
$Side = Buy$	$Side = Buy$	$Side = Buy$
$CP = CP_1$	$CP = CP_1$	$CP = CP_1$
$ExP = P_1$	$ExP = P_1$	$ExP = P_1$
$ExS = S_1$	$ExS = S_2$	$ExS = S_3$
$TV = TV_1$	$TV = TV_1$	$TV = TV_1$
$S = RT$	$S = RT$	$S = RT$
$TInt = TI_1$	$TInt = TI_1$	$TInt = TI_1$
$Alert = 1$	$Alert = 1$	$Alert = 1$

According to these results, it is clear that some details of the dataset and the model need to be considered in the future work.

TABLE IV.

Precision	**Recall**	**F-measure**
1.000	0.662	0.797

VII. Conclusions and future work

This paper presents the following conclusions:

1) The analysis of transactions extracted from an electronic trading protocol allowed us to successfully create a model for the classification of market abuse behavior.
2) The proposed scenarios allowed to test the model more thoroughly. The following checks were included: consistency checks, checks for border values, checks for empty values, etc.
3) The prototype testing revealed some weaknesses that manifested themselves through unexpected behavior in the case of empty values, in the case of border values (for some of the attributes) and also in the case of different values for one of the categorical attributes.

As focus of our future research, we propose to generate the test cases using the equivalence classes methodology. We advise that the equivalence classes be set according to the types of data (categorical, numeric, etc.) and their border values. Such an approach enables us to parametrise the equivalence classes and to further develop an automatic tool that generates the test data.

References

[1] FCA (financial conduct authority): Available at: https://handbook.fca.org.uk/

[2] SEC (Securities and Exchange Commission): Available at: https://www.sec.gov/

[3] FINMAR Financial Stability and Market Confidence Sourcebook:Available at: https://handbook.fca.org.uk/handbook/FINMAR/

[4] Cao L., Ou Y., Yu P.: Detecting Abnormal Coupled Sequences and Sequence Changes in Group-based Manipulative Trading Behaviors. In Proc. of KDD10, Washington, DC, USA, July 2528, 2010, 85–93

[5] Donoho S.: Early Detection of Insider Trading in Option Markets In Proc. of KDD04, Seattle, Washington, USA, August 2225, 2004, 420–429

[6] Luo C., Zhao Y., Cao L., Ou Y., Zhang C.:Exception Mining on Multiple Time Series in Stock Market.In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, 690–693

[7] Nasdaq and Digital Reasoning Establish Exclusive Alliance to Deliver Holistic Next Generation Surveillance and Monitoring Technology:Available at: http://www.digitalreasoning.com/buzz/nasdaq-and-digital-reasoning-establish-exclusive-alliance-to-deliver-holistic-next-generation-surveillance-and-monitoring-technology.1884035

[8] Ou Y., Cao L., Luo C., Liu L.:Mining Exceptional Activity Patterns in Microstructure Data.In Proc. of International Conference on Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM, 2008, 884-887

[9] Ou Y., Cao L., Yu T., Zhang C.:Detecting Turning Points of Trading Price and Return Volatility for Market Surveillance Agents.In Proc. of International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops, IEEE/WIC/ACM, 2007, 491–494

[10] Murphy C., Kaiser G., Arias M.:An Approach to Software Testing of Machine Learning Applications.Proc of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE), Boston MA, Jul 2007, 167–172

[11] Nautiyal L, Preeti:A Novel Approach of Equivalence Class Partitioning for Numerical Input.ACM SIGSOFT Software Engineering Notes. Volume 41 Issue 1, 2016, 1–5

[12] Murphy C., Kaiser G., Arias M.:Parameterizing Random Test Data According to Equivalence Classes.Proc of the 2nd International Workshop on Random Testing (RT'07), Atlanta GA, Nov 2007, 38–41

[13] Murphy C., Kaiser G., Arias M.:A Framework for Quality Assurance of Machine Learning Applications.Columbia University Computer Science Technical Reports, New York, 2006

[14] Murphy C., Kaiser G., Hu L., Wu L.:Properties of Machine Learning Applications for Use in Metamorphic Testing.Proc of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE), Redwood City CA, Jul 2008, 867-872.

[15] Zhang J., Wang Z., Zhang L., Hao D., Zang L., Cheng S., Zhang Lu.:Predictive Mutation Testing.In Proc. of ISSTA16, Saarbrcken, Germany, July 1820, 2016, 342–353

Initial steps towards assessing the usability of a verification tool

Mansur Khazeev*, Victor Rivera† and Manuel Mazzara‡
Innopolis University, Software Engineering Lab.
Innopolis, Russia
Email: *m.khazeev@innopolis.ru, †v.rivera@innopolis.ru, ‡m.mazzara@innopolis.ru

***Abstract*—In this paper we report the experience of using AutoProof to statically verify a small object oriented program. We identified the problems that emerged by this activity and we classified them according to their nature. In particular, we distinguish between tool-related and methodology-related issues, and propose necessary changes to simplify both tool and method.**

I. Introduction

Formal proof of correctness of the software is still not usual thing nowadays, even though the hardware and the software technologies for verification has significantly improved since it was first mentioned about verifying compiler - a cipher for an integrated set of tools checking correctness in a very broad sense [1]. In ideal world, verifying software would need only pushing a button, though this kind of provers exist, but they can verify only simple or implicit properties such as absence of invalid pointer dereference [2]. However to be able to fully verify a software, a specification, like contracts in design by contract methodology, should be provided against which it will be verified. The term was introduced in connection with design of the Eiffel programming language, but nowadays it is adopted for many other popular languages. There is a prover for functional correctness of programs written in Eiffel language called AutoProof. Having a powerful methodology for framing and class invariants, it fully supports advanced object-oriented features [2]. In order to test the usability and the ability of the tool to be applied in general practice, a series of case studies are being conducted, verification of ordinary class, a subset of classes of Tokeneer system [3] and Tokeneer system entirely by non-expert in the field. This paper describes the results of the first step - verification of class MY_SET, that implements classic sets from set theory, and its classical operations.

The challenge of this exercise is mainly related to difficulties of the tool usage. There is no explicit documentation available for the users: only the website as a main source, and several papers from the authors of the the tool. However the notation was evolving and in some cases these papers are not up relevant. As it was previously said verification with AutoProof often requires additional annotation that help the tool to derive the more complex properties from trivial ones. However, for someone who does not know how the tool works and what is going on under the hood, the feedback from the tool would not always be helpful. Of course, for some specific tool, which is to be used by a narrow group of scientists this is excusable, but since the main purpose is to make the tool for static verification applicable in industrial practice - complete documentation should be developed in the meanwhile minimizing the need of additional assertions. It is essential because the tool still requires a knowledge of underneath mechanisms and a number of extra annotations.

II. AutoProof

AutoProof is a static verifier for programs written in Eiffel, which is a real complex object oriented programming language that natively supports Design-by-Contract methodology [4]. Users can specify the behavior of Eiffel classes by equipping them with contracts: pre- and post-conditions and class invariants, that are represented as assertions [3]. This allows to minimize amount of specification and effort needed to reason about programs properties. AutoProof follows the auto-active paradigm where verification is done completely automated, similar to model checking [5], but users are expected to feed the classes providing additional information in the form of annotations to help the proof. The tool is capable to identify software issues without executing the code, therefore opening a new frontier for "static debugging", software verification and reliability, and in general for software quality.

AutoProof verifies the functional correctness of the code written in Eiffel language equipping with contracts. The tool checks that routines satisfy pre- and post-conditions, maintenance of class invariants, loops and recursive calls termination, integer overflow and non **Void** (*null* in other programming languages) references calls. For that AutoProof uses an intermediate verification language Boogie: it translates the Eiffel code into a file and feeds it to a tool called Boogie [6] as well. The Boogie tool generates verification conditions (logic formulas whose validity entails correctness of the input programs) that are then passed to an SMT solver Z3. Finally, the answer retrieved back to Eiffel.

The tool supports most of the Eiffel language constructs: inlined assertions such as **check** (*assert* in other programming languages), types, multi-inheritance, polymorphism.

By default AutoProof only verifies user-written classes when a program is verified, referenced libraries should be verified separately or should be based on preverified library *base_eve*. [7]. This preverified library has most of abstract classes, the naming starts with prefix *V_* meaning that it is verified [3].

III. Case study experience

This case study is a small exercise which was to implement in Eiffel programming language a generic class MY_SET using V_LINKED_LIST class and equipping it with contracts. On the next step this class had to be proved for correctness adding all needed annotations.

Basically, the exercise was about expressing some properties of the set as invariants:

- No duplicating elements
- Order of elements in the set does not matter
- Cardinality is always bigger or equal to 0

And implementing some basic operations:

- is_empty - set with no elements in it
- cardinality - number of elements in the set
- has - if the set contains some specific element
- is_strict_subset, is_ subset - if the set is subset of other set
- union/intersection/difference - functions returning new set from two other

During the process of verification, it turned out, that for non-expert users working with V_ classes was too complicated, therefore the decision was done to ease the example replacing V_LINKED_LIST with SIMPLE_LIST.

IV. Problems taxonomy

Despite the class under study was rather simple, it resulted in facing many different problems because of lack of experience with the AutoProof tool before. Beginning with issues of the tool installation and kept up till checking the verified class with tests. While analyzing the results, all those problems were divided into two main categories: problems with the tool and with the approaches or methodologies used in the tool. The first category includes rather minor problems and bugs, mostly related to the particular implementation in the tool and means that those require local fixes. However, the second category require improving the methodology or replacing them with the alternative ones.

A. Problems with the tool

The challenge of this exercise was mainly related the the fact that it had to be done by a person who has no previous experience with AutoProof, nor any other similar tools. The difficulty is not in some sophisticated user interface (UI) of the tool, which is by the way, rather simple - a ”Verify” button and a table (see figure 1), where the results are being displayed. The main obstacle is in the fact that, the tool expects an input in terms of assertions, and it is not always clear what the real effect of each input is.

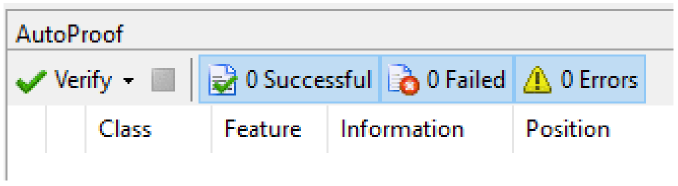


Fig. 1. UI of AutoProof

1) Lack of documentation: As it was previously said the tool requires additional annotations that lead the verification and help to derive one property from another. Although AutoProof uses the syntax of Eiffel language, there are many new things has been introduced by the developers of the tool. Most of this briefly described in an on-line manual which is available on the web-site of EVE[1]. In addition there is an on-line tutorial which are useful for quick acquaintance with the tool. However, this is surely not enough for using it as an reference while working with the tool.

Overall, there is not much of documentation available in the Internet: the website, and several papers from the authors of the AutoProof tool, however the notation is some of these papers are not relevant anymore. Of course, for some specific tool, which is to be used by a narrow group of scientists this is excusable, but if we are talking about applying verification in industrial practice - the documentation would be essential because the tool still requires quite a bit of additional annotations and knowledge of underneath mechanisms.

2) Feedback from the tool: The process verification (or static debugging) starts with pushing a ”Verify” button and when the tool returns some feedback (error and failure messages) - fixing them by adding required assertions and repeating the procedure until everything is ”successful”. A failure message is a message that describes the property that can not be satisfied. The error messages returned by the tool are information on some issue with the input. AutoProof implements a collection of heuristics known as two-step verification that helps discern whether failed verification is due to real errors or just to insufficiently detailed annotations [2]. The failure messages usually informative, they describe the property and sometimes may describe what can be the reason. However, the error messages usually do not tell more that that the tool could not proceed with input it has received. It occurs due to problems in ”translation”.

If there is a failure during translations between AutoProof and Boogie the verification process stops and the AutoProof returns an error message about ”internal failure” in some cases with no additional information. Usually this can be resolved only if the error was caused by newly entered assertion, because otherwise it is not possible to learn what exactly causes this error. This may become an issue while verifying a class with all features implemented and contracts stated because it is not possible to determine the source of the the

[1]EVE (Eiffel Verification Environment) - is a development environment integrating formal verification with the standard tools

error. The solution might be to add features one by one, verifying each one separately.

3) Redundancy in notations: AutoProof supports most of the Eiffel language as used in practice [2], as well as introduces some new notations to support the methodologies used for verification. Usually those notations are useful for decreasing amount of assertion by skipping repetitions, for manipulating the defaults of semantic collaboration in the feature or the class (will be discussed in IV-B1).

However, some of these additional notations introduce redundancy. For example, the Eiffel class, containing one or more creation procedures should have these procedures explicitly listed under keyword **create**. In figure 2 **make** is defined as creation procedure. However, the verifier expects additional **note** clause with **status: creator** in order to treat it as creation procedure.

```
create
  make

feature
  make
    note
      status : creator
    do
      ...
```

Fig. 2. Denoting a procedure **make** as creation

Another example is depicted in the figure 3 were the status of the function has to be defined with **impure** keyword, meaning that it modifies the state of the object, and then, **modify** clause with empty squire brackets that stands from function purity. This is done just to be able to use **wrap/unwrap** in the function.

4) Misleading notations: AutoProof support inline assertions and assumptions, which can be expressed using **check** clause. **Checks** are intermediate assertions that are used during the debugging process to check if you have the same understanding of the state at a program point as the verifier [8]. However, removing an intermediate **check** clause from successfully verified feature might fail later verification, because **check** assertions guide the verifier towards the successful verification. Perhaps the developers of the tool made the decision not to add new keyword and to use **check** clause from Eiffel in order to keep thing simpler. However, this might convey users to a wrong understanding of the impact of the clause.

5) Order of assertions: The **check** clauses might be useful because the verifier not just checks the property enclosed, but also uses it in further derivations, in case it was proven correct. Same applies to invariants, and that makes order of properties substantial for the tool. Basically, this means that properties are joint not by **and** operator but **and then** instead, which may lead to verification failures even if all needed properties are stated but in improper order.

6) Limitations of the tool: Currently, AutoProof has some limitation on usage of the verified library eve-base. Even though, this library is fully verified, it can only be used when the void-safety property of the tool is disabled. The latest works demonstrate that this library was checked for void safety, however this versions are not available yet.

7) UI bug: The tool lacks of support which can be observed in some rare bugs. For example it can skip some of the features of the class or verify only one of the feature instead of whole class. Even though, the tool never returned improper successful verification results, this kind of bugs might be annoying.

8) Compilation from the sources: The tool is still raw and it is better not to use the latest versions. The repository requires some clean up, because currently, if one try to build the verifier from some last sources, it just will not compile.

B. Methodology: Problems with Semantic collaboration

AutoProof supports advanced object-oriented features and a powerful methodology for framing and class invariants, which make it applicable in practice to idiomatic object-oriented patterns [2]. But this power comes with price of simplicity - the tool requires understating of all these methodologies under the hood. This makes the tool available for the expert users only by over complicating the verification of even a relatively simple classes.

1) Semantic collaboration: AutoProof supports semantic collaboration, a full-fledged framing methodology that was designed to reason about class invariants of structures made of collaborating objects [2]. This methodology introduces own annotations which does not existed in Eiffel language. Annotations are used to equip features and classes entirely with additional information which proceeded by the verifier. These include many "ghost" attribute [2] which are quite useful when maintenance of global consistency is required as in subject/observer or iterator pattern examples [9].

Meanwhile, all these ghost attributes and default assertions that are added into pre- and post-conditions unreasonably over-complicate verification process of rather simple classes.

In the earlier stages of verification quite a bit of time was spent trying to understand the failure message that **default_is_closed** may be violated on calling some feature" of some private attribute **data**. Basically, the tool was expecting **owns = [data]** in the invariants of the class which is not obvious without understanding the methodology. Moreover, for this specific example the property could be derived from exportation status of the attribute. The verifier ignores this useful informations and requires the properties stated explicitly. Eiffel language supports the notion of "selective export", which exports the features that follow to the specific classes and their descendants [10]. Considering this information might narrow the need for using semantic collaboration [11].

2) Framing: The framing model is used in AutoProof in order to help reason about objects that are allowed/not allowed to be updated. There are different ways to specify

[2]class members used only in specifications

this by adding modifies clauses in preconditions: one can specify one or more model fields, attributes of the class or list of objects which may be updated. This is rather intuitive and straightforward, though seems to be more relevant to postcondition clauses. There are default clauses included into each routine, so it should be used only if behavior of the routines are different from default. In MY_SET example all the routines had to be pure and because of that modify clause had to be specified in each of them. Even in a function, which is pure by default, if one wishes to use "is_wrapped" clause, that function needs to be specied as impure and in a meanwhile, that it does not modify anything (see figure 3); which looks confusing.

```
feature -- Queries
  union(other : likeCurrent) : likeCurrent
    -- New set of values contained in 'Current' or 'other'
  note
    status : impure
  require
    modify_nothing : modify([ ])
    ...
```

Fig. 3. Pure function **has** specified as impure

V. Related Work

Formal notations to specify and verify software systems have existed for long. A survey of the major approaches can be found in [12], while [13] discusses the most common methodological issues of such approaches.

Design-by-contract [14] combined with static verification technologies could offer wide applicability in practice. Nowadays, many popular programming languages support this methodology using embedded contract languages [15], and some are expected to support it on a language level soon.

In [16] the authors present an extensive survey of algorithms for automatic static analysis of software. The discussed techniques (static analysis with abstract domains, model checking, and bounded model checking) are different, but complementary, to the one discussed in this paper, and they are also able to detect programming errors or prove their absence.

The importance of focusing on usability requirements for verification tools has been identified in [17]. In particular, the non-expert usage of AutoProof has been studied in [18] where programmers with little formal methods experience have been exposed to the tool.

VI. Conclusion

AutoProof is not trivial in usage and needs detailed knowledge of what is going on behind the scenes. The tool requires a number of additional assertions in pre- and post-conditions, as well as in invariants for successful verification, ignoring some information that has been already provided. To be used in practice the usability of the tools should be significantly improved making verification simple enough to be used by ordinary programmers. By simple we mean, that it should:

- require less additional annotations by: automatically deriving properties from information which currently is neglected; removing redundant clauses and reworking some of ghost class members;
- provide clearer feedback in case some property can not be satisfied, offering hints and possible solution;

In addition, it is important to:

- develop a documentation describing all used methodologies, including detailed information about notations with examples
- clean up and rebuild the tool from latest sourced that are available in EVE repository, fixing all the bugs that we identified;

References

[1] J. Woodcock, E. G. Aydal, and R. Chapman, *The Tokeneer Experiments*, pp. 405–430. London: Springer London, 2010.

[2] J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, "Autoproof: Auto-active functional verification of object-oriented programs," *CoRR*, vol. abs/1501.03063, 2015.

[3] M. Khazeev, V. Rivera, M. Mazzara, and A. Tchitchigin, "Usability of autoproof: a case study of software verification," in *Proceedings of the Institute for System Programming, vol. 28, issue 2*, pp. 111–126, 2016.

[4] J. Tschannen, C. A. Furia, M. Nordio, and N. Polikarpova, "Autoproof: Auto-active functional verification of object-oriented programs," in *21st International Conference, TACAS 2015, London, UK, April 11-18, 2015. Proceedings*, pp. 566–580, 2015.

[5] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[6] K. R. M. Leino, "This is boogie 2," tech. rep., June 2008.

[7] "Autoproof manual." [Online] http://se.inf.ethz.ch/research/autoproof/manual/ (Date last accessed: 2016-03-15).

[8] E. Z. Chair of Software Engineering, "Autoproof tutorial,"

[9] C. A. F. Nadia Polikarpova, Julian Tschannen and B. Meyer, *Flexible Invariants through Semantic Collaboration*, pp. 514–530. Cham: Springer International Publishing, 2014.

[10] B. Meyer, *Touch of Class: Learning to Program Well with Objects and Contracts*. Springer Publishing Company, Incorporated, 1 ed., 2009.

[11] D. de Carvalho, "Modularly reasoning in object-oriented programming using export status." unpublished, 2017.

[12] M. Mazzara and A. Bhattacharyya, "On modelling and analysis of dynamic reconfiguration of dependable real-time systems," in *2010 Third International Conference on Dependability*, pp. 173–181, 2010.

[13] M. Mazzara, "Deriving specifications of dependable systems: toward a method," in *Proceedings of the 12th European Workshop on Dependable Computing, EWDC*, 2009.

[14] B. Meyer, *Object-oriented software construction*, ch. 11: Design by Contract: building reliable software. Prentice Hall PTR, 1997.

[15] M. Fähndrich, M. Barnett, and F. Logozzo, "Embedded contract languages," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, (New York, NY, USA), pp. 2103–2110, ACM, 2010.

[16] V. D'Silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Trans. on CAD of Integrated Circuits and Systems*, no. 7, pp. 1165–1178.

[17] R. Razali and P. Garratt, "Usability requirements of formal verification tools: A survey," *Journal of Computer Science*, no. 6, pp. 1189–1198.

[18] C. A. Furia, C. M. Poskitt, and J. Tschannen, "The AutoProof verifier: Usability by non-experts and on standard code," in *Proceedings of the 2nd Workshop on Formal Integrated Development Environment (F-IDE)* (C. Dubois, P. Masci, and D. Mery, eds.), vol. 187 of *Electronic Proceedings in Theoretical Computer Science*, pp. 42–55, EPTCS, June 2015. Workshop co-located with FM 2015.

The Study into Cross-Site Request Forgery Attacks within the Framework of Analysis of Software Vulnerabilities

Alexander Barabanov
Testing Department
NPO Echelon
Moscow, Russia
ab@cnpo.ru

Artem Lavrov
Testing Department
NPO Echelon
Moscow, Russia
mail@cnpo.ru

Alexey Markov
Information Security Department
Bauman MSTU
Moscow, Russia
a.markov@bmstu.ru

Ivan Polotnyanschikov
Testing Department
NPO Echelon
Moscow, Russia
mail@npo-echelon.ru

Valentin Tsirlov
Information Security Department
Bauman MSTU
Moscow, Russia
v.tsirlov@bmstu.ru

Abstract— **This work presents a study into security of web-applications, which are the targets of evaluation within the framework of certification as to information security requirements, against cross-site request forgery attacks. It specifies results of classification and summary of information about similar attacks and different types of defense against them. It specifies results of the study, which demonstrate distribution of identified vulnerabilities as per the developer type, distribution of the protective measures used in web-applications, distribution of the identified vulnerabilities as per the programming languages, data on the number of protective measures that are used in the studied web-applications. The results of the study showed that in the majority of cases the developers of web-applications does not pay due attention to protection against cross-site request forgery attacks. The work gives recommendations to the developers that are planning to undergo certification process for their software.**

Keywords— ***information security; software security; analysis of vulnerabilities; web-application; CSRF-attack***

I. INTRODUCTION

Software created with the use of web-technologies is currently one of the main components in automated control system (ACS) design. The designed ACS are, as a rule, multi-user and can be found on public domain networks (for instance, Internet), which increases the risk of their successful attack. Various procedures (such as certification, independent security audit) are currently used to lower probability of successful attack. They are aimed at identifying vulnerabilities in the software used to design ACS [1, 2].

Software vulnerabilities are analyzed during certification for compliance with the requirements to the protection profiles approved by FSTEC of Russia (Federal Service for Technology and Export Control), which clearly includes requirements of AVA_VAN assurance family “Vulnerability analysis”, and during testing for compliance with the requirements of the technical specifications and classic governing documents of FSTEC of Russia. The procedure for vulnerability analysis recommended by FSTEC of Russia consists in the joint use of approaches specified in the Common Methodology for Information Technology Security Evaluation and ISO/IEC TR 20004 [3]. It should be noted that more specific instructions for the test laboratories (for instance, standard penetration tests) have not yet been developed, which makes this procedure non-determined [4].

The experience of analysis into vulnerabilities of web-applications within the framework of the accredited test laboratory showed that Cross-Site Request Forgery attack, hereinafter – CSRF-attack is currently the most successful attack against targets of evaluation. The main attention of the developers of web-applications, as a rule, is concentrated on implementing measures protecting against attacks like SQL-injections or Cross-site scripting. The situation is aggravated by the fact that measures protecting against CSRF-attacks are still being actively studied, and best practices have not been rigidly registered yet [2, 6].

The goal of this work consisted in developing guidelines for the developers of web-applications, who are planning to certify their solutions as to the information security requirements. The work solves the following tasks to achieve the set goal:

a) Classification and summary of information about CSRF-attacks and measures of protection against them;

b) Consolidation of information about vulnerabilities of web-applications identified within the framework of work of the accredited test laboratories.

II. The results of classification and summary of information about CSRF-attacks and relevant security measures

A hacker performing a CSRF-attack makes the web-browser used by the legal user, who has been authenticated in "security measures against " the attacked web-application, send HTTP-request, which is going to be identified by the application as a request received from a legal user, to the web-application.

A possible consequence from a successful CSRF-attack implementation is running of an arbitrary code in the web-application in the name of authenticated user. Thus, the main causes of CSRF-attacks are vulnerabilities in web-applications related to wrong implementation of algorithm of HTTP-request authorization. Success of CSRF-attack is determined by the following factors [7, 8]:

- The browser automatically applies authentication data of the user (for instance, session cookie-files), when sending HTTP-request to the web-application;
- Web-application uses the obtained authentication data to authorize the action required for performance by HTTP-request.

It should be noted that despite difficulties in implementation, there are cases of successful CSRF-attacks of ‘Login’ and ‘Logout’ type on web-applications [1, 9, 10]. The probability of successful ‘stored’ CSRF-attack is higher, because a malicious code is stored on the side of the attacked web-application, and the hacker does not have to make the user (for instance, using methods of social engineering) go to a special resource with a malicious code.

Implementation of the security measures on the client’s side [11-16], represented by plugins/extensions of the browser or additional software (proxy), has significant drawbacks [8] and is currently only of academic interest.

There are suggestion on implementing security measures directly with the browser source code, for instance, using ‘samesite’ properties of the cookie-files, but currently these measures are experimental and are implemented only in certain browsers. Integrated measures (measures implemented jointly by the software code on the client- and the server-sides), as a rule, implement a certain information control policy [6, 17], which contain critical information (for instance, authentication data), between the browser and the web-server. It should be noted that effective implementation of this type of security measures is possible by making changes in the browser source code. Moreover, essential limitations of these security measures are well-known, which does not allow their use as a sole measure of protection.

The most popular security measures against CSRF-attacks are tokens (synchronic tokens or generated using HMAC cryptographic function) that are generated and checked on the web-application side. This security measure is implemented, as a rule, by the web-application itself or the framework. It should be noted that the majority of the most popular frameworks (such as, Ruby on Rails, ASP.NET, Django) implement this measure, which somewhat decreases the workload for the developer of a certain web-application and reduces the number of errors related to implementation of the security algorithm by the developer of the web-application.

The main distinctive feature of the token-based security measures is in the token storage method:

- Generated token may be stored on the web-application side (it is associated with the user session) and it shall be compared with the token received from the web-browser;
- Generated token may be stored on the web-browser side (for instance, in the cookie); when the web-application receives a request from the web-browser, the web-application compares the values of tokens in the cookie and the HTTP-request body.

It should be noted that this measure of the web-application security is used correctly, if it is designed and implemented in a way that HTTP-requests of GET type do not change the server state, and are used only for request of the necessary information. AJAX-requests may be protected with tokens inserted in HTTP-header, or custom HTTP-headers (during implementation of this security measure the web-application only checks availability of the heading in the received request).

The leading specialists in the web-application security recommend using the defense in depth principle, when implementing security measures. Thus, specialists of OWASP community recommend implementing security of the web-application by combining two types of the security measures – HTTP-headers verification and tokens.

In some cases, the developers use three or more security measures for critical information systems (for instance, online banking systems). For example, it can be a combination of tokens, verification of HTTP-header and security measures that require actions from the end user, who performs a critical operation (entry of one-time code/ password).

III. Methods and Results of the Study

The study into the security level of the web-application was carried out in the accredited test laboratory of NPO Echelon (study period: January – November 2016). Brief information about the web-applications that participate in the study is represented in Table 1.

TABLE I. BRIEF INFORMATION ABOUT THE STUDY OBJECTS

Software identifier	**Programming language**	**Type of developer**	**Level of measures for secure software development implementation (maturity level)**
Software No. 1	PHP	Russian	2
Software No. 2	Java	Foreign	5
Software No. 3	PHP	Russian	1
Software No. 4	Java	Foreign	5
Software No. 5	C#	Russian	4
Software No. 6	Java	Russian	1
Software No. 7	C#	Russian	1
Software No. 8	PHP	Russian	1
Software No. 9	Ruby	Russian	3
Software No. 10	Ruby	Russian	3

Level of measures for secure software development implementation (maturity level) was assessed by the expert method with account of the scope of measures implemented by the developer of measures from the basic set of measures for developing secure software suggested in the National Standard GOST R 56939-2016 Information Protection. Secure Software Development. General Requirements. [4, 18]: *1* - not one measure is implemented, *2* - less than 20% of measures is implemented, *3* – from 20% to 40% of measures is implemented, *4* - from 40% to 60% of measures is implemented, *5* - from 60% to 80% of measures is implemented, *6* - over 80% of measures is implemented.

Vulnerabilities were analysed using standard tests developed with account of recommendations and CAPEC resource. Below is the general sequence of the performed tests:

1) Analysis of parts of web-applications (pages), which allow changing the state of the web-application (creating/ changing/ deleting user accounts, protected information, other information etc.).

2) Study of the requests to the identified parts of web-applications: transmission of the requests from the web-browser to the web-application with further interception and analysis of the request structure. The expert analyses the intercepted request and defines the type of security measure against CSRF-attack on a specific page.

3) Generating a mock HTTP-request, which is saved as an HTML-file on the local computer and is opened in the web-browser, provided that there is a session authenticated by the target of evaluation (web-application).

4) If the analysis of intercepted request (cl. 2) revealed security measures against CSRF-attacks, the following actions shall be additionally taken:

a) When tokens are used as a security measure:

- Analysis of URL for a presence of token in a plain text;
- Sending a request without a token;
- Sending a request with an altered token;
- Sending a request using one token for various user accounts;
- An attempt to guess /select a token;

b) When using verification of the HTTP-headers as a security measure:

- Sending a request with altered HTTP Referer (originally a misspelling of «referrer»)/Origin fields;
- Sending a request without HTTP Referrer/Origin fields.

The tests were performed using the following software: BurpSuite software, Scaner-VS software. The average time spent on testing of one web-application by one expert of the test laboratory is 8 hours.

The results of the study are specified below.

1) CSRF-attacks were successful in 70% of cases – 7 out of 10 analysed web-applications turned out to be vulnerable.

2) The majority of CSRF-attacks were successful in relation to web-applications developed in Russia. It should be noted that the only CSRF-attack that was successful in relation to the foreign web-application was that of "Logout" type, and the experts of the test laboratory failed to develop an attack vector that implements information security threat. Only one web-application initially did not have any security measures against CSRF-attacks. The other vulnerable web-applications had security measures based on verification of HTTP-headers or token (Figure 1).

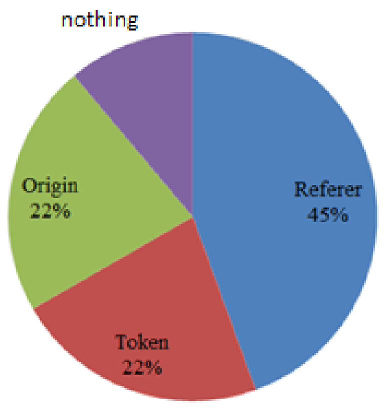


Fig. 1. Distribution of protection measures used in vulnerable web-applications

3) It has been established that web-applications written in PHP have a few more vulnerabilities that results in successful CSRF-attacks (Figure 2) [20].

4) The developers upgraded vulnerable web-applications using security measures based on tokens in all cases.

5) In the majority of cases the upgraded web-application and web-applications, where the vulnerability has not been identified, used a combination of several security measures against CSRF-attacks.

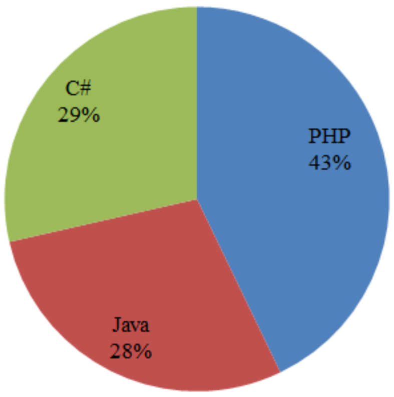


Fig. 2. Distribution of identified vulnerabilities as to the programming language

6) The average time required for the web-application developer to correct vulnerability is 3 weeks.

7) One of the results of the study was a deduced empirical rule, in accordance with which the number of vulnerabilities identified in the software is in inverse proportion to the maturity level of the secure software development processes implemented by the developer.

IV. Recommendations to Developers on Increasing the Security Level of Web-Applications

Based on the results of the study the following recommendations were provided for the developers of web-applications that are planning to hold certification tests as to information safety requirements.

1) It is advisable that the developers implement measures for secure software development in the software lifecycle processes. At the very least, it is recommended to implement measures related to testing penetration of web-application prior to their submission to the test laboratory. To minimize time for such testing, the developers should generate sets of standard tests, which may be developed with account of guidelines represented in the works [17, 19]. The developers are advised against limiting their tests to the standard test only, and are recommended to run additional tests aimed at performing CSRF-attacks, like ‘Login’ and ‘Logout’, and verify that the selected security measure is correctly implemented.

2) The developers are recommended using the defense in depth principle – combine two or more security measures (as a rule, verification of token and HTTP-headers), when implementing security measures against CSRF-attacks in the web-application.

3) When implementing security measures against CSRF-attacks in the web-application, the developers are first of all recommended to use security measures that are already implemented in the operational environment, for instance, frameworks.

V. Conclusions

This work consisted in the study into security of web-applications, which are the test targets within the framework of certification as to information security requirements, against cross-site request forgery attacks. The result showed that the majority of the developers (around 70%) do not pay due attention to implementing security measures against such attacks. Resulting from the study, we defined recommendations for the developers, the main of them being recommendations on the use of defense in depth principle and the use of token-based security measures that had already been implemented by the framework developers. We deduced empirical rule, in accordance with which the number of vulnerabilities identified in the software is in inverse proportion to the maturity level of the secure software development processes implemented by the developer. Further studies are intended into the issues of the web-application protection against SQL-injection attacks and cross-site scripting attack and defining general guidelines for the developers of web-applications, who are planning certification.

References

[1] H. Selim, S. Tayeb, Y. Kim, J. Zhan, and M. Pirouz. “Vulnerability Analysis of Iframe Attacks on Websites,” In Proceedings of the The 3rd Multidisciplinary International Social Networks Conference on SocialInformatics 2016, Data Science 2016 (MISNC, SI, DS 2016). ACM, New York, NY, USA, Article 45 , pp. 1-6, August 2016. DOI: 10.1145/2955129.2955180.

[2] W. Du, K. Jayaraman, X. Tan, T. Luo, and S. Chapin. “Position paper: why are there so many vulnerabilities in web applications?” In Proceedings of the 2011 New Security Paradigms Workshop (NSPW '11). ACM, New York, NY, USA, pp. 83-94. 2011. DOI: 10.1145/2073276.2073285.

[3] A. Barabanov, A. Markov, A. Fadin, V. Tsirlov, I. Shakhalov. “Synthesis of Secure Software Development Controls,” In Proceedings of the 8th International Conference on Security of Information and Networks (Sochi, Russia, September 8-10, 2015). SIN '15. ACM, New York, NY, USA, pp. 93-97. 2015. DOI: 10.1145/2799979.2799998.

[4] A.V. Barabanov, A.S. Markov, V.L. Tsirlov. “Methodological Framework for Analysis and Synthesis of a Set of Secure Software Development Controls,” Journal of Theoretical and Applied Information Technology. 2016. V. 88. No 1, pp. 77-88.

[5] N. Jovanovic, E. Kirda, and C. Kruegel. “Preventing cross site request forgery attacks,” In the IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (Securecomm) , pp. 1–10, September 2006.

[6] A. Czeskis, A. Moshchuk, T. Kohno, and H.J. Wang. “Lightweight server support for browser-based CSRF protection,” In Proceedings of the 22nd international conference on World Wide Web (WWW '13). ACM, New York, NY, USA, 2013, pp. 273-284. DOI: 10.1145/2488388.2488413.

[7] K. Jayaraman, P. G. Talaga, G. Lewandowski, S.J. Chapin, and M. Hafiz. “Modeling user interactions for (fun and) profit: preventing request forgery attacks on web applications,” In Proceedings of the 16th

Conference on Pattern Languages of Programs (PLoP '09). ACM, New York, NY, USA, , Article 16 , pp. 1-9. August 2009. DOI: 10.1145/1943226.1943246.

[8] A. Barth, C. Jackson, and J.C. Mitchell. “Robust defenses for cross-site request forgery,” In Proceedings of the 15th ACM conference on Computer and communications security (CCS '08). ACM, New York, NY, USA, pp. 75-88. October 2008. DOI: 10.1145/1455770.1455782.

[9] M. Zhou, P. Bisht, and V.N. Venkatakrishnan. “Strengthening XSRF defenses for legacy web applications using whitebox analysis and transformation,” In Proceedings of the 6th international conference on Information systems security (ICISS'10), Somesh Jha and Anish Mathuria (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 96-110. 2010.

[10] E. Shernan, H. Carter, D. Tian, P. Traynor, and K. Butler. “More Guidelines Than Rules: CSRF Vulnerabilities from Noncompliant OAuth 2.0 Implementations,” In Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148 (DIMVA 2015), Magnus Almgren, Vincenzo Gulisano, and Federico Maggi (Eds.), Vol. 9148. Springer-Verlag New York, Inc., New York, NY, USA, pp. 239-260. June 2015. DOI: 10.1007/978-3-319-20550-2_13.

[11] H. Shahriar and M. Zulkernine. “Client-Side Detection of Cross-Site Request Forgery Attacks,” In Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10). IEEE Computer Society, Washington, DC, USA, pp. 358-367. November 2010. DOI : 10.1109/ISSRE.2010.12.

[12] P.D. Ryck, L. Desmet, T. Heyman, F. Piessens, and W. Joosen. “CsFire: transparent client-side mitigation of malicious cross-domain requests,” In Proceedings of the Second international conference on Engineering Secure Software and Systems (ESSoS'10), Fabio Massacci, Dan Wallach, and Nicola Zannone (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 18-34. 2010. DOI: 10.1007/978-3-642-11747-3_2.

[13] R. Pelizzi and R. Sekar. “A server- and browser-transparent CSRF defense for web 2.0 applications,” In Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11). ACM, New York, NY, USA, pp. 257-266. December 2011. DOI: 10.1145/2076732.2076768.

[14] L. Xing, Y. Zhang, and S. Chen. “A client-based and server-enhanced defense mechanism for cross-site request forgery,” In Proceedings of the 13th international conference on Recent advances in intrusion detection (RAID'10), Somesh Jha, Robin Sommer, and Christian Kreibich (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 484-485. 2010.

[15] N. Gelernter and A. Herzberg. “Cross-Site Search Attacks,” In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, pp. 1394-1405. October 2015. DOI: 10.1145/2810103.2813688.

[16] E. Z. Yang, D. Stefan, J. Mitchell, D.Mazières, P.Marchenko, and B. Karp. “Toward principled browser security,” In Proceedings of the 14th USENIX conference on Hot Topics in Operating Systems (HotOS'13). USENIX Association, Berkeley, CA, USA, pp. 17-17. 2013.

[17] W. Maes, T. Heyman, L. Desmet, and W. Joosen. “Browser protection against cross-site request forgery,” In Proceedings of the first ACM workshop on Secure execution of untrusted code (SecuCode '09). ACM, New York, NY, USA, pp. 3-10. November 2009. DOI: 10.1145/1655077.1655081.

[18] A. Barabanov, A. Markov, V. Tsirlov. “Procedure for substantiated development of measures to design secure software for automated process control systems,” In Proceedings of the International Siberian Conference on Control and Communications, SIBCON 2016, IEEE, 1-4. June 2016. DOI: 10.1109/SIBCON.2016.7491660.

[19] X. Li and Y.Xue. “A survey on server-side approaches to securing web applications,” ACM Comput. Surv. 46, 4, Article 54 (March 2014), 29 pages. April 2014. DOI: 10.1145/2541315

[20] A.S. Markov, V.L. Tsirlov. “Opyt vyyavleniya uyazvimostey v zarubezhnykh programmnykh produktakh”, Voprosy kiberbezopasnosti [Cybersecurity Issues]. 2013. No 1(1), pp. 42-48. (In Russ.).

Dataflow Analysis for the Search of the Code Security Defects

Sergei Borzykh
Development Department
NPO Echelon
Moscow, Russia
mail@cnpo.ru

Alexey Markov
Information Security Department
Bauman MSTU
Moscow, Russia
a.markov@bmstu.ru

Andrei Fadin
Development Department
NPO Echelon
Moscow, Russia
af@cnpo.ru

Valentin Tsirlov
Information Security Department
Bauman MSTU
Moscow, Russia
v.tsirlov@bmstu.ru

Pavel Gusev
Development Department
NPO Echelon
Moscow, Russia
mail@npo-echelon.ru

***Abstract*—This paper is devoted to the static analysis of the source code security and use of the dataflow analysis algorithms for effective identification of code defects (errors made during development), and intentional backdoors. The paper discusses implementation of this algorithm and shows how it can be used for identification of potentially harmful structures in the source code.**

***Keywords*— information security; software security; static analysis; heuristic analysis; vulnerabilities; defects; production models; data-flow analysis**

I. Introduction

IT-based solutions are currently used everywhere, and significant problems are represented by both internal software errors of the information systems, and malicious source code implemented in the information system software. The consequences of both problems lead to violation of access, integrity and confidentiality of the processed information, which can result in financial and reputational losses of the business. This is a reason of growing financial losses over the last few years. High quality and failure-free operation of the source code is a burning issue of the software industry. Ever growing complexity of the software complexes, their use in the management and control systems of the government and the industrial production require continuous upgrading of the software testing and control methods [1-9].

II. Static and Dynamic Methods of the Code Analysis

Upon the whole, the testing methods used in the audit of the software systems security may be divided into two groups: static methods (structural testing) and dynamic methods (functional testing).

Static methods of the code analysis, which do not require running of the analysed code for its operation, allow for full or partial automation [10, 11]. Such methods are most frequently used in case of full access to the software system and its source texts, which is called “a white-box technique”. It employs source and loading modules of the program and its component. The benefit of the static code analysis is that it does not require multiple program runs under various operational conditions (condition of the environment and input data) and possibility to achieve a greater degree of automation of the tests for the program defects based on their design features. When developing software for special-purpose informational systems, these methods are used to search for random code defects, and hidden software functionality (backdoors) [12, 13].

Dynamic software analysis is a method of analysis that stipulates program running on real or virtual processor [14]. Functional testing is most in demand during the study of the programs by black box method, when there is access to only external software interfaces without account of their structure, back-end interfaces or status. The approach is used to study accuracy and stability of the software operation within the framework of the key jobs of the test engineers, however, the method is not always effective for searching of errors related to combinations of rarely used input data, and for identifying intentional backdoors there.

Static analysis of the software source texts is closely related to development of compilation systems, and many approaches of static analysis use elements of the compiler theory, namely, the code view models [15, 16].

III. Signature Analysis as the Main Method

The approach that is called signature analysis implies the search for software defects in the software code by comparing code fragments with the samples from the database of templates (signatures) of the security defects. Depending on the method for correlating fragments of the code to the template, and the intermediate representation in use, there may be algorithms of searching for a substring in the string, and query language for structured information (for instance, XQuery for XML), or specially designed methods of correlation, but in each case each of the signatures represents the decision procedure, which employs various presence bits of potentially harmful structure. [16] provides examples of the rules for generating error signatures, which correspond to the CWE standard. We can see here that the signature methods are not limited to the types of defects and are preferable, when dealing with the backdoors.

Improvement of the operational qualities of the static code analysis is mainly related to minimizing the number of "false positives" while preserving maximum fullness of the list of the types of potentially harmful structures [17]. Therefore, the instruments describing signatures of the code defects shall ensure maximum flexibility in defining a defect with account of diversity in the syntax of the programming language under study.

The field for designing means of static analysis is now actively developing: new directions of analysis do not force out the reputable approaches, on the contrary, they complement them by integrating the advantages of the predecessors. For instance, such approach as dataflow analysis may compensate for the drawbacks of the template-based code defect search, which does not allow for high quality of identification of SQL-, Path-, XSS-injections, and other types of code injections, however, it will require large RAM and computing resources of the processor [18, 19, 20].

An interesting manifestation of symbiosis of the analysis methods is when potentially harmful structures, which have been initially identified by the customary signature method is supported by the automated method using highly-specialized, costly, but efficient procedures [21-23].

IV. Dataflow analysis

The dataflow analysis can be described as a process of gathering information about the use, defining and dependency of data in the analysed program [24, 25]. The dataflow analysis uses command flow graph generated based on the code tree. This graph represents all possible paths for running this program: the nodes stand for 'linear', consecutive fragments of the code without any transitions, and the edges stand for potential transfer of control between these fragments.

Syntactic analysis allows for identifying control structures, such as procedure, function or method calls, which, in their turn, allow building call graphs, control flow graphs, and identifying assignation and the others that allow building dataflow graphs [15, 16, 26]. Control and dataflow graphs are used for analysis of the local program blocks (mainly, the content of the functions, procedures and methods - local analysis). Control flow graphs allow analysing program behaviour on a more general level (on the level of the file, module or the entire program - global analysis).

The dataflow analysis can be used for proper detection of certain types of defects (as a rule, in operation) with a minimum number of false positives: SQL-, command-, XSS-injections, other types of code injections and setting directly in the code of the authentication data. It should be noted that despite the differences in these defects, most of them implement the following defect use pattern.

1. Data is received from the user (consequently, untrusted data).
2. Data propagates through the program depending on the conditions and cycles.
3. Data is transformed, or filtered, or remains unchanged.
4. Finally, untrusted data gets access to the vulnerable function (buffer management, SQL query running etc.).

There is a mechanism for dataflow analysis called "taint propagation", which allows for identifying the defect, but also shows the data propagation path, starting from the entry point (user input), through the program and to the function vulnerability [27]. An interesting instance of such mechanism of dataflow analysis is "constant propagation" - search for authentication data (login, password, IP-address) directly in the software source code. Let us review a code fragment:

```
String login = "Some Constant";
```

Such code fragment can be sought using signature analysis (search as per templates). It only requires representation rule:

```
VARIABLE ("login" OR "password")
OPERATOR ("=") CONSTANT(*);
```

However, these code fragments can show that such code was written for debugging and remained in the final software version by accident, or was added intentionally, provided there was assurance that the code would not be inspected. If a malicious developer wants to hide the imbedded defect from the person, who inspects the code, but also from the means of static analysis, the code may be written, for instance, this way:

```
   String label = "somewhere".substring(0,4);
   String summ =
LogConstant.class().getClassName().toLowerCase()
;
   String upd_time = summ.substring(3,
summ.lenght()-3);
   Char ascii_conv = 95;
   String login = label + ascii_conv + upd_time;
```

If we break down parts of code fragment into various modules and files of source texts, it will be next to impossible to identify the defect using manual analysis, as well as many known automated methods.

The "constant propagation" mechanism of the dataflow analysis may define the values of the variables, their concatenation and transfer into other variables, and final values of the variables. As a result, the defect may be identified and, consequently, unauthorized access to the functional capabilities of the software may be prevented.

V. Operating principles and application of the dataflow analysis

Previous sections show the importance of static analysis and general issues. Following sections describe the main approach for dataflow analysis implementation and results of its implementation in static analyzer AppChecker developed by NPO Echelon. Let us introduce a set of definitions for a future shorter description of the principles and algorithms of this method operation:

- Point — a node in the control flow graph;
- Touch points (TP) (sink or critical points) — nodes in the control flow graph, which are used for calling important functionality (in the context of the identified defect);
- Entry point - nodes in the control flow graph, where new data is received from interfaces outside the analysed code;
- Untrusted data - data received from interfaces outside the analysed code and trusted zone (allied agents, users);
- Critical flow - flow from the entry point to the touch point.

Let us define the general procedure for the search for undocumented features using dataflow analysis:

1. Prepare source texts and configurations of the analysed software.
2. Use of the static analysis tools (that implement dataflow analysis) to sourced texts and configurations prepared in step 1.
3. Processing of the results of analysis:
    - Selecting suspicious dataflow paths,
    - Analysing entry points and points of untrusted data propagation,
    - Filtering false positives.
4. Drawing up the final report.

Dataflow analysis is divided into two stages. The first stage of analysis requires engineering of critical control and dataflows in the analysed software. Below is the sequence of the algorithm actions.

1. Search for the entry points of the untrusted data in the analyzed software (template-based search). This step requires a base of entry points templates formed by inspections of standard libraries and popular frameworks.
2. Search for the points that contain potentially vulnerable functions (template-based search as well).
3. For each entry point of the untrusted data, add function, method and procedure calls that are happening in this point to the control flow tree.
4. Repeat clause 3 until you reach one of the final points specified in clause 2, or until you reach a point that does not transition into other functions, procedures or methods.
5. Once control flow trees are built, identify flows that have reached potentially vulnerable functions. Consider these flows critical.

At the second stage, analyse critical control flows, their separate points (functions, procedures and methods) and identify the fact of untrusted data propagation from the entry point to the potentially vulnerable function. Below is the sequence of the algorithm actions:

1. Obtain entry point (function, procedure or method), engineer all dataflows that affect the data received from untrusted source.
2. If untrusted data after interaction with other dataflows has not changed its status, proceed to clause 3. Otherwise, complete analysis of the current critical control flow.
3. If untrusted data were transmitted at the following point of the critical control flow, proceed to clause 4. Otherwise, complete analysis of the current critical control flow.
4. If the current point is the endpoint, proceed to clause 5. Otherwise, proceed to clause 2 with a new point and new input data. Continue, until analysis of all points in the critical control flow is complete.
5. If the current point is the endpoint, and untrusted data were transmitted to the potentially vulnerable function from the first point, enter the critical control flow on the positive triggering list. Otherwise, finalize analysis of the current critical control flow.

The list obtained at the entry to the second stage of analysis is transferred to the entry of the report generator, which control interface is also present within the graphical user interface; after that the report generator based on the transferred list and database of the defect types draws up a report on the performed static analysis.

VI. Local analysis

The following description refers to dataflow implementation in static analyzer AppChecker developed by NPO Echelon. The local analysis is normally performed for a certain block of the code (which coincides with the visibility scope depending on the programming language). The local analysis assumes obtaining information about the conditions of the program in all points of the program, i.e.:

- On creating data;
- On saving data;
- On destruction of data.

The diagram of the local analysis algorithm can be seen in Figure 1.

You can optionally store, for instance, data on the value constancy (for the “constant propagation” tool), data assurance flag (for “taint propagation”), and information about the condition of the variable.

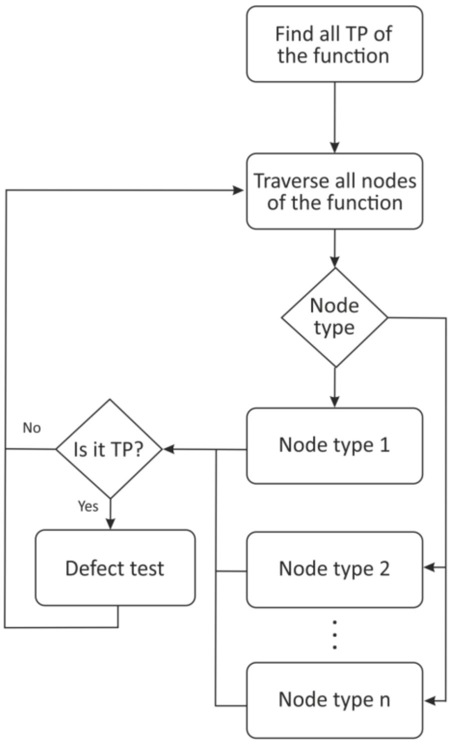


Fig. 1. Diagram of the Local Analysis Algorithm.

Information can be obtained from the local block in two opposite ways listed below:

1. From bottom to top: from the point susceptible to the defect make assumptions about the properties of the data transmitted into it, go up the code to the point of entry in the local area (function, procedure or method). The approach requires consideration of all options of the program run (for each branch and iteration of the cycles), which leads to “combinatorial explosion” of the information quantity, which shall be stored during analysis.

2. From top to bottom — from the point of entry of untrusted data into the program, down along the code, with available information about all of the above points of the program. This approach allows making assumptions about running separate branches of the program and engineer sequential analysis. The drawback of this approach is difficulty in obtaining the path from the entry point to the exit point, because the only known fact is that the path exists.

VII. GLOBAL ANALYSIS

Information that is available within one function (procedure, method), as a rule, is insufficient for high quality search for the defects, because many defects propagate throughout the project, or, at least one file. Global analysis is used to link data received from different functions. The global analysis engages call graphs. To ensure operation of this analysis it is sufficient to obtain certain confirmation or assumption about the properties of input and output data of separate functions in the call graph. It is important to obtain such data in the context of the functions, which are outside of the path from the point of the data entry to the point susceptible to the defects, and which analysis is necessary because the call of such functions may change the arguments or return values, which properties may depend on the properties of the input data (for instance, the substring get function, which accepts data input by the user returns taint data, although formally it is not included in the call graph).

The diagram of the global analysis algorithm can be seen in Figure 2.

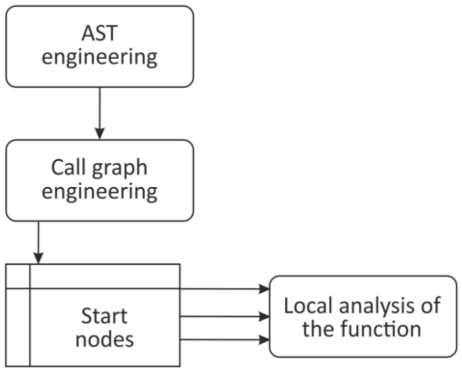


Fig. 2. Diagram of the Global Analysis Algorithm.

VIII. MATHEMATICAL DESCRIPTION OF THE DATAFLOW ANALYSIS

The ideal solution of the dataflow analysis task from the theoretical point of view consists in the search for all possible paths. Let us introduce certain symbols: B — data block for analysis, which consists from elementary subblocks $B_1, \ldots, B_n$. It is a known fact, that the dataflow values before the statement and after it are limited by the semantics of the instruction. The correlation between the dataflow values before and after the assignment statement is characterized by the transfer function. f_i shall stand for a transfer function of block B_i, which characterizes transformation of data in this block. The values of the dataflow before and after subblock B_i shall be represented as IN[B_i] (OUT[B_i] accordingly).

Suppose P is a possible execution path in the flow graph:

$$P = \text{Input} \rightarrow B_1 \rightarrow \ldots \rightarrow B_k.$$

In this case, the transfer function f_P for path P will be represented by a composition of the transfer functions $f_{k-1} \bullet \ldots \bullet f_1$. However, it should be noted that f_k is not a part of the composition, which shows that the path reached the start of subblock B_k, but not its end. Let us consider that any flow graph consists of two empty subblocks - input block, which is a start point of the graph, and output block, which is passed by all exits from the graph. The transfer functions of input and output blocks are represented by constant values.

Thus, taking into account the foregoing, the ideal solution is the array:

$$IDEAL(B) = \bigcup_P f_P(v_{input}),$$

where v_{input} is the result of the constant transfer function, which is represented by the starting input node.

It may seem that the task of the search for the ideal solution is reduced to analysis of the transfer functions f_P for all paths P in the flow graph. However, it was noted by Ullmann [15, page 724], the task of the search for the ideal solution is generally unsolvable. If block B has branches, cycles and recursions, array IDEAL[B] maybe unlimited. The assistance comes from the solution of path-based gathering [15, page 757], which is similar to the path search algorithm in the graph, so called 'breadth first search'. This algorithm allows achieving such final number of P, that an array of all f_P covers all unique transformations of f_B.

Let us write down an iterative solution to the generalized task for the dataflow. There are two versions of such algorithm - direct and reverse. The first version proceeds from input blocks to the output, the second - goes in the reverse. The basis is Ullmann's algorithm [15, page 754].

Direct version of the algorithm:

```
OUT[INPUT] = $v_{input}$;
For (each base block B, which differs from input)
    OUT[B] = InitDataConst;
while (changes are entered in OUT)
    for (each basic block B, which differs from input) {
 IN[B] = $\bigcup_{P\text{-predecessor}}$ OUT[P] ;
    OUT[B] = $f_B$ (IN[B]);
    }
```

Reverse version of the algorithm:

```
IN[INPUT] = voutput;
for(each basic block B, which differs from output)
    IN[B] = InitDataConst;
while (changes are entered in IN)
    for(each basic block B, which differs from output) {
    OUT[B] = $\bigcup_{P\text{-predecessor } B}$ IN[P] ;

    IN[B] = $f_B$ (OUT[B]);
    }
```

Subject to [15], if algorithm converges, its result is the solution to the dataflow problem. The obtained solution turns out to be a so called maximum fixed point, which has the property that in any other solution *IN[B]* and *OUT[B]* are already present in this solution. If in this case the analysed block is final, the convergence of the algorithm is guaranteed. These statements are proved by Jeffrey Ullmann in [15, pages 754-755].

It should be noted that in practice it is inadvisable to analyse all data used by the program. For example, if we consider unfiltered user input as input data vinput, we are going to be interested in B, where *OUT[B]* are entered in the database or output in HTML-context. Block B may also be represented by the function of the input information filtering, thus finalizing the path and marking it as safe.

IX. EXAMPLES AND RESULTS

Dataflow analysis is widely spread in compilers [15] and some sort of program analysis tools in order to find mistakes, typos and other accidentally inserted source code errors or weaknesses. This paper is dedicated to the implementation and usage of well known analysis approach for detecting potentially harmful code areas deliberately inserted into source code. The paper subject novelty is in joint usage dataflow and signature template-based analysis for detection both embedded malicious code (backdoors, trapdoors, hard-code credentials) and weaknesses caused by accidental developer's mistakes.

Below are the examples of potentially harmful structures detected by the method described here using AppChecker software. These examples are real but quite simple because we think it is unacceptable to provide big and complex examples in this article.

1. Potential SQL-injection is identified in Dolibarr project, in *htdocs/admin/menus/edit.php* file:

B_{284} = «`$sql = "SELECT m.rowid, m.mainmenu, m.level, m.langs FROM ".MAIN_DB_PREFIX."menu as m WHERE m.rowid = ".$_GET['menuId'];`»

B_{285} = «`$res = $db->query($sql);`»

Data received from the user is entered in $sql variable, and the value of the variable without filtration is entered in SQL-request, which may lead to running of random SQL code. The critical point is string B_{285}; constant string is concatenated with taint data, and as a result the part of the string to the right of concatenation becomes taint.

2. The use of passwords set directly in the software code is identified in AWCM project, in *connect.php* file:

B_3 = «`$db_hostname = "localhost";`»

B_4 = «`$db_username = "root";`»

B_5 = «`$db_userpass = "123456";`»

B_6 = «`$db_database = "awcm";`»

B_{24} = «`@mysql_connect($db_hostname, $db_username, $db_userpass);`»

Parameters, including the password, set directly in the code, are used to connect to the database. The critical point is string B_{24}; in string B_5 the right part of the expression is a constant; in practice, the string is allocation of constant value to the variable used further to set the password.

Nowadays AppChecker, which implements algorithms of signature analysis using flow analysis, contains the total of 253 rules for the search of defects in the software code in four programming languages: C/C++, Java, PHP, C#; the rules allow identifying 113 types of defects [26, 28]. AppChecker was tested in 90 projects with open source codes.

X. Conclusion

The following conclusions can came from the results of the study:

1. Based on well-reputed signature analysis approach, the suggested method of the dataflow analysis can minimize the number of false positives and simplify the development of signatures for an analyser production model.

2. The suggested method and tools will be useful for the accredited testing laboratories as well as developers of safe software tools. Secure software development practices (we would, first of all, like to mention a recently approved national standard in this field [29]), are being implemented at a growing rate nowadays, therefore integration of the structured testing procedure in the process of the automated system development based on static signature analysis is a high-priority task.

References

[1] D.Yu. Volkanov, V.A. Zakharov, D.A. Zorin, V.V. Podymov, I.V. Konnov, "A Combined Toolset for the Verification of Real-Time Distributed Systems," Program. Comput. Softw., vol. 41, no. 6, pp. 325-335, November 2015. DOI:10.1134/S0361768815060080.

[2] I. S. Zakharov, M. U. Mandrykin, V. S. Mutilin, E. M. Novikov, A. K. Petrenko, and A. V. Khoroshilov, "Configurable toolset for static verification of operating systems kernel modules," Program. Comput. Softw., vol. 41, no. 1, pp. 49-64, January 2015. DOI: 10.1134/S0361768815010065.

[3] I. S. Anureev, I.V Maryasov, and V.A. Nepomniaschy, "C-programs verification based on mixed axiomatic semantics," Autom. Control Comput. Sci., vol. 45, no. 7, pp. 485-500, 2011. January 2012.

[4] P. N. Devyanin, A. V Khoroshilov, V. V Kuliamin, A. K. Petrenko, and I. V Shchepetkov, "Formal Verification of OS Security Model with Alloy and Event-B BT - Abstract State Machines, Alloy, B, TLA, VDM, and Z: 4th International Conference, ABZ 2014, Toulouse, France, June 2-6, 2014. Proceedings," Y. Ait Ameur and K.-D. Schewe, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 309-313.

[5] D. Beyer and A. K. Petrenko, "Linux Driver Verification BT - Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies: 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part II," T. Margaria and B. Steffen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1-6.

[6] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko, "Pair Programming and Software Defects--A Large, Industrial Case Study," IEEE Transactions on Software Engineering, vol. 39, no. 7. pp. 930-953, Jul. 2013.

[7] S. M. Avdoshin and E. Y. Pesotskaya, "Software risk management," 2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR). pp. 1-6, 2011.

[8] A. S. Kamkin and M. M. Chupilko, "Survey of modern technologies of simulation-based verification of hardware," Program. Comput. Softw., vol. 37, no. 3, pp. 147-152. May 2011.

[9] G. Reber, K. Malmquist, A. Shcherbakov. "Mapping the application security terrain," Voprosy kiberbezopasnosti [Cybersecurity Issues], No 1, pp. 36—39. January 2014. (In Russ).

[10] A. Cox, B.-Y.E. Chang X. Rival. "Automatic Analysis of Open Objects in Dynamic Language Programs," International Static Analysis Symposium, Static Analysis, pp. 134-150, September 2014. DOI: 10.1007/978-3-319-10936-7_9.

[11] G. Balatsouras, Y. Smaragdakis. "Structure-Sensitive Points-To Analysis for C and C++", International Static Analysis Symposium, Static Analysis, pp. 84-104, September 2016. DOI: 10.1007/978-3-662-53413-7_5.

[12] F. Zhu, J. Wei. "Static analysis based invariant detection for commodity operating systems," Computers and Security, vol. 43, pp. 49-63, June 2014. DOI: 10.1016/j.cose.2014.02.00.

[13] M. Bradley, F. Cassez, A. Fehnker, T. Given-Wilson, R. Huuck. "High performance Static Analysis for Industry," Electronic Notes it Theoretical Computer Science, vol. 289, pp. 3-14, December 2012. DOI: 10.1016/j.entcs.2012.11.002.

[14] P. Gonzalez-de-Aledo, P. Sanchez, R. Huuck. "An Approach to Static-Dynamic Software Analysis," Proceedings of International Workshop on Formal Techniques for Safety-Critical Systems, pp. 225-240, November, 2015. DOI: 10.1007/978-3-319-29510-7_13.

[15] A.V. Aho, M.S. Lam, R. Sethi J.D. Ullman. Compilers: Principles, Techniques, and Tools (2nd Edition). Addison Wesley; 2nd edition (September 10, 2006).

[16] A.S. Markov, A.A. Fadin, V.L. Tsirlov. "Multilevel Metamodel for Heuristic Search of Vulnerabilities in the Software Source Code," International Journal of Control Theory and Applications. V. 9. N 30, pp 313-320, December 2016.

[17] M. Junker, R. Huuck, A. Fehnker, A. Knapp. "SMT-based false positive elimination in static program analysis," Proceedings of 14th International Conference on Formal Engineering Methods, Japan, Volume 7635 of LNCS. Springer, pp. 316-331, November 2012. DOI: 10.1007/978-3-642-34281-3_23.

[18] W. Choi, S. Chandra, G. Necula, K. Sen. "SJS: A Type System for JavaScript with Fixed Object Layout," International Static Analysis Symposium, Static Analysis, pp. 181-198, September 2015. DOI: 10.1007/978-3-662-48288-9_11.

[19] Z. Luo, T. Rezk, M. Serrano. "Automated code injection prevention for web applications," Proceedings of the 2011 international conference on Theory of Security and Applications, pp. 186-204, March 2011. DOI: 10.1007/978-3-642-27375-9_11.

[20] D. Ray, J. Ligatti. "Defining code-injection attacks," Proceeding of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 179-190, January 2012. DOI: 10.1145/2103656.2103678.

[21] S. Seo, A. Gupta, A. Sallam, E. Bertino, K. Yim. "Detecting mobile malware threats to homeland security through static analysis," Journal of Network and Computer Applications, vol: 38 (1) pp. 43-53, February 2014. DOI: 10.1016/j.jnca.2013.05.008.

[22] W. Lee, H. Oh, K. Yi. "A Progress Bar for Static Analyzers," International Static Analysis Symposium, Static Analysis, pp. 184-200, September 2014, DOI: 10.1007/978-3-319-10936-7_12.

[23] E. Goubault, S. Putot, F. Vedrine. "Modular static analysis with zonotopes," International Static Analysis Symposium, Static Analysis, pp. 24-40, September 2012. DOI: 10.1007/978-3-642-33125-1_5.

[24] P. Calvert, A. Mycroft. "Control Flow Analysis for the Join Calculus," International Static Analysis Symposium, Static Analysis, pp. 181-197, September 2012. DOI 10.1007/978-3-642-33125-1_14.

[25] M. Madsen, A. Moller. "Sparse Dataflow Analysis with Pointers and Reachability," International Static Analysis Symposium, Static Analysis, pp. 201-218, September 2014. DOI: 10.1007/978-3-319-10936-7_13

[26] A. Markov, A. Fadin, A. Shvets, V. Tsirlov. "The experience of comparison of static security code analyzers," International Journal of Advanced Studies, vol. 5. № 3. pp. 55-63, September 2015. DOI: 10.1109/MS.2008.130.

[27] D. Zhu, J. Jung, D. Song, T. Kohno, D. Wetherall "TaintEraser: protecting sensitive data leaks using application-level taint tracking," Newsletter ACM SIGOPS Operating Systems Review archive, January 2011, Volume 45, Issue 1, pp. 142-154. DOI: 0.1145/1945023.1945039.

[28] A.S. Markov, V.L. Tsirlov. "Opyt vyyavleniya uyazvimostey v zarubezhnykh programmnykh produktakh", Voprosy kiberbezopasnosti [Cybersecurity Issues]. 2013. No 1(1), pp. 42-48. (In Russ.).

[29] A.V. Barabanov, A.S. Markov, V.L. Tsirlov. "Methodological Framework for Analysis and Synthesis of a Set of Secure Software Development Controls," Journal of Theoretical and Applied Information Technology. V. 88. No 1, pp. 77-88, June 2016.

Debugger for Real-Time OS: Challenges of Multiplatform Support

Alexander Emelenko
Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russian Federation
emelenko@ispras.ru

Kurban 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*—Debugger for a real-time OS is an important tool in software development process. However, debugger's code has to be developed for each platform. We faced the problem of porting our debugger to different architecture without developing it from scratch. In this paper, we present the architecture of the debugger for JetOS real-time operating system and discuss the challenges imposed by multiplatform support in the OS.**

***Keywords*— debugger, operating systems, multiplatform**

I. Introduction

Application debugger is an indispensable tool in developer's hands. But debugger in a real-time operating system is more than just plain debugger. In this paper we present an on-going project on debugger development for JetOS, a real-time multiplatform operating system that is being developed in the Institute for System Programming of the Russian Academy of Sciences.

JetOS is a prototype operating system for civil airborne avionics. It is designed to work within Integrated Modular Avionics (IMA) architecture and implements ARINC-653 API specification, the de-facto architecture for applied (functional) software.

The primary objectives of ARINC 653 are deterministic behavior and reliable execution of the functional software. To achieve this ARINC-653 imposes strict requirements on time and space partitioning. For instance, all memory allocations and execution schedules are pre-defined statically.

The unit of partitioning in ARINC-653 is called partition. Every partition has its own memory space and is executed in user mode. Partitions consist of one or more processes, operating concurrently, that share the same address space. Processes have data and stack areas and they resemble well-known concept of threads.

Embedded applications might be run in two different environments: in an emulator and on the target hardware. In our project we use QEMU system emulator. Although QEMU has its own debugger support, its functionality proved to be insufficient for debugging embedded applications. Therefore we implemented a debugger not only for the target hardware, but for the emulator as well.

II. Specific Features of Debugger for RTOS

Developing a debugger for a real-time OS is not a simple task. During developing, we faced many challenges of the debugger for RTOS compared to typical debuggers used by desktop developers.

Firstly, an embedded solution might contain a number of interacting processes, which need to be debugged simultaneously. Therefore, our debugger must support capability to switch between them. Moreover, it needs to support work with overlapping virtual addresses space.

Secondly, it is impossible to run the debugger on the same target system, where the system runs, because of the lack of on-board resources and lack of interactive facilities. That's why the debugger must be remote.

Thirdly, we must support debugging not only for application developers but also for system software developers, such as drivers or kernel developers. As a consequence, the debugger should work with both a privileged kernel and low-privilege application code.

Also, the debugger must support debugging on both target hardware and emulators, because it can expand developers' capabilities and increase their efficiency.

Moreover, JetOS runs on different CPU families: it supports PowerPC, x86 and MIPS architectures. Consequently, the debugger has to run on these platforms too. Thus, the debugger must consider all features of all platforms and emulators, and provide full functionality and correct execution of applications.

We base our debugger on GDB (GNU debugger) architecture. It includes uniform CPU-independent user interface on the developer's workstation and target-specific remote stub that implements the GDB protocol to receive commands from the user interface and provide it back with registry data, memory contents and status updates.

In this paper we present the experience gained in GDB stub debugger and discuss the ways to respond to the identified challenges.

III. Related Works

We are not the first to consider the problem of debugging multiplatform RTOS. There are many types of debuggers: some of them utilize pure hardware facilities without any code injected into the target system, such as CodeWarrior, others use remote debugging, for example, the debugger for Pistachio microkernel; besides, there is RTOS debugger for VxWorks that supports both scenarios.

Here we briefly consider a number of debuggers for embedded OSes and their primary features.

A. CodeWarrior

CodeWarrior [3] is an IDE (integrated development environment) published by Freescale Semiconductor. It is designed to edit, compile and debug software for several microcontrollers and microprocessors (Freescale ColdFire, ColdFire+, Kinetis, Qorivva, PX, Freescale RS08, Freescale S08, and S12Z) and digital signal controllers (DSC MC56F80X and MC5680XX) used in embedded systems. It uses JTAG or BDM interface to control the target system.

CodeWarrior enables the user to debug real-time embedded applications, as well as manipulate the source code to display and change the contents of variables, arrays, and data structures. The developer can also use the debugger to work at the hardware level if necessary.

Via CodeWarrior user can:

- View and change memory, registers and variables.
- Set watchpoints.
- Set breakpoints and conditional breakpoints.
- Break on exceptions.
- Track variables

The core of the CodeWarrior debugger is a dedicated iMX-6 based microcomputer, that connects with one end to JTAG plug on the target and with the other end – to the developer's workstation. The microcomputer interprets the JTAG signals in the real time to track execution: reading/writing of CPU registers, memory mapped registers, block reading/writing of memories, single step debugging, setting software and hardware breakpoints, and monitoring target system status.

CodeWarrior provides GDB interface to the target system without injecting any code to the embedded software, but it works for Freescale (now NXP) boards only and requires accesible JTAG connector.

B. VxWorks

VxWorks [5] is a real-time operating system (RTOS) developed as proprietary software by Wind River of Alameda, California, US. It supports Intel (x86, including the new Intel Quark SoC and x86-64), MIPS, PowerPC, SH-4, and ARM architectures.

The debugger for WindRiver Workbench provides a rich set of features. It implements various transport channels between the target system and the workstation. It support debugging in emulators, remote debugging using on-board software agents and by means of JTAG probes.

The debugger provides a general interface called On-Chip Debugger API so that third party IDE or developer scripts can automate debugging tasks.

Here is an incomplete list of features that the debugger for VxWorks implements :

- Task Stack Coverage
- Task Related Breakpoints
- Task Context Display
- Debugging Modules (for example, Kernel module)
- Debugging Real-Time Processes
- Debugging Protection Domains
- Collecting statistics for function and tasks

The debugger can control all system states, tasks, message queues, memory partitioning, modules and etc. in real time.

C. L4Ka::Pistachio

L4Ka::Pistachio [4] is an L4 microkernel developed by the System Architecture Group at the University of Karlsruhe. It is the first available kernel implementation of the L4 Version 4 kernel API, which provides support for both 32-bit and 64-bit architectures, multicore, and superfast local IPC. The current release supports x86-x64 (AMD64/ EM64T, K9 / P4 and higher), x86-x32 (IA32, Pentium and higher), PowerPC 32bit (IBM 440, AMCC Ebony / Blue Gene P).

Pistachio kernel uses kdbg debugger. The debugger directs its I/O via the serial line or the keyboard/screen. It is a local debugger and does not support remote debugging mode, therefore it has a very limited amount of functions.

Debugger for Pistachio can:

- Set breakpoints
- Single step
- Dump memory
- Read registers

Debugger for L4Ka::Pistachio supports two platforms, x86 and PowerPC. It is implemented by dividing debugger's code into platform specific and independent parts.

Architecture dependent part of the debugger includes:

- Registers printing
- Single step support
- Memory writing
- TLB printing
- Breakpoints setting

IV. Debugger Architecture for Multiplatform Support

Our debugger consists of two parts – server and client. We don't support JTAG probes for now.

We use GDB for the client part of our debugger and it has the biggest part of architecture independent code. For example, the client part has such operations as disassembly, messaging mechanism, user interface, etc.

In general, messaging mechanism between client and server doesn't depend on the CPU family of the target – user communicates with the client, the client sends a special-type packet to the server and waits for the server's answer. The server receives this message, checks control sum, which was sent in this packet, and if it matches the message contents, informs the client that the message was accepted for processing. Then the server performs the action described in the packet and sends its own packet to the client. Understanding of what the client wants from the server is a part of architecture independent code in OS because almost all client requests are standardized, but their execution depends on current hardware.

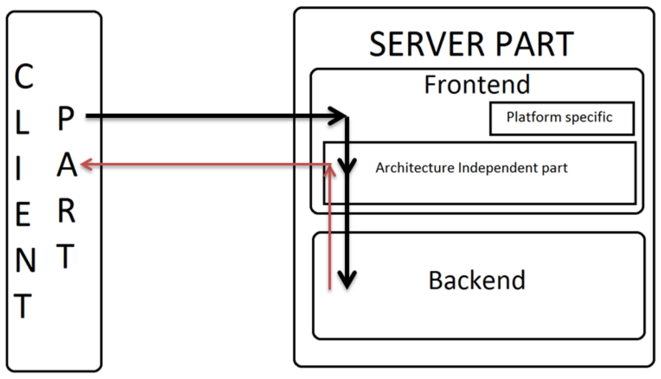


Fig. 1. Debugger communication

We can divide our server part of the debugger into 2 parts as shown in Fig.1:

A. *Frontend*

This part parses packets, checks control sum, calls the needed function and sends a reply.

Although almost all client-server packets are architecture independent, such requests as registers read/write depend on the target hardware. Therefore, our parser must know not only which architecture is used, but also know the type of packet the client wants to receive: for example, if the client wants to read all registers, the server must send 70 registers on PowerPC and 72 on MIPS.

B. *Backend*

This part of the debugger considers all platform capabilities and uses all available resources.

The largest part of target specific code is responsible for setting breakpoints, watchpoints, single step and read/write in memory. To implement such features not only we do need specialized code in server part, but we must also change exception handler so that it could distinguish between debugger and regular interrupts.

For example, the single step function is implemented in PowerPC architecture via special debug register, used only in this processor. Moreover, breakpoint function needs to set trap instruction where the user wants: for x86 architecture, this is ‘int3’ instruction, for MIPS – ‘break’ instruction.

The option to debug applications not only on actual hardware metal but also in emulators, such as QEMU, is another our goal. Due to partial implementation of hardware features in QEMU the GDB stubs designed for the target hardware just don’t work in QEMU. For example, there are no debug registers in QEMU for PowerPC architecture, which is used for single step realization on bare metal. Consequently, for each hardware target we provide two GDB server implementations – one for bare metal and another one for QEMU.

V. Debugger’s Capabilities

As mentioned above, all debugger's capabilities are available for all supported platforms – x86, PowerPC and MIPS.

A. *Setting Breakpoints in Partitions and Kernel*

Control execution of partitions and kernel is a key feature of debugging. It provides capabilities to more adapted debugger control mechanisms. Moreover, our system supports work with overlapping virtual address spaces, which means that debugger must correctly translate it into physical address.

B. *Execute the Application Step-by-Step*

Run application step by step is a convenient way to control system state and finding bugs. However, next instruction in code might not be the next executable instruction, for example, because of interrupt. Therefore, user can choose to stop on next instruction in code via disabling interrupts or on next executable instruction via platform capabilities.

C. *Inspect the Application State*

In each moment of time, user might want to inspect system state, i.e. memory, registers, stack trace, and threads state.

D. *Setting Watchpoints*

Watchpoints provide a great opportunity to control system state. The developer can use watchpoints to stop execution whenever the value of an expression changes, without having to predict a particular place where this may happen.

VI. Debugger’s Portability

As already mentioned, JetOS is being developed now, so we don’t know the final set of platforms which our system will have to support.

To port our debugger on a new platform, we need only to change the CPU-specific part of the frontend and create the backend part. All other frontend parts are the same for all CPUs and platforms.

It is easy to predict the amount of effort needed to port the debugger to a new platform with the help of these values:

In PowerPC server part consists of over 2000 lines of code:

- Over 1700 loc – frontend part, of which
  - 1600 – architecture independent part
  - 100 – platform specific part
- Over 300 – backend part

This separation provides capabilities for porting our debugger to a new platform in little to no time.

VII. Conclusion

The paper presents a multiplatform debugger for JetOS real time operating system. It is based on GDB client-server architecture with most of the CPU-specific code located in the server (on-board) part. Due to careful design the on-board part of the debugger clearly separates platform-independent code of the protocol processor and general debug control algorithms from platform-specific operations such as reading the registers, setting breakpoints and performing single-step instructions.

The amount of platform-specific code in the on-board part of the debugger comprises 25% of the overall debugger server, thus making porting of the debugger to another platform relatively simple task.

References

[1] Lauterbach GmbH, "RTOS debugger for VxWorks", November 2015 http://www2.lauterbach.com/doc/rtosvxworks.pdf

[2] Lauterbach GmbH, "RTOS-VxWorks", 18 August 2014 http://www2.lauterbach.com/pdf/rtos_vxworks.pdf

[3] Freescale Semiconductor, Inc. CodeWarrior Debugger, December 2, 2004 http://www.nxp.com/assets/documents/data/en/reference-manuals/Engine_PPCRM.pdf

[4] System Architecture Group University of Karlsruhe. "The L4Ka::Pistachio Microkernel". May 1, 2003 http://www.l4ka.org/l4ka/pistachio-whitepaper.pdf

[5] Wind River Systems, Inc "VxWorks Product Overview", March 2016 http://www.windriver.com/products/product-overviews/VxWorks-Product-Overview-Update.pdf

[6] Free Software Foundation, Inc. "Debugging with gdb: the gnu Source-Level Debugger", The Tenth Edition

Using modularization in embedded OS

K. Mallachiev, N. Pakulin, A. Khoroshilov, D. Buzdalov,
Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia.

***Abstract*— Modern embedded OS are designed to be used in control solutions in various hardware contexts. Control computers may differ in the architecture of the CPU, the structure of communication channels, supported communication protocols, etc. Therefore, RTOS should support modularity and component assembly in order to allow customers to create minimal solutions that are optimally adapted to the particular task and hardware platform. Furthermore customers may be interested in adding their own low level components without OS modification.**

In this article, we present an approach to building modular embedded solutions from heterogeneous components based on the RTOS JetOS. The mechanism of components binding developed by us allows uniting heterogeneous components from different manufacturers within the same section of the address space.

***Keywords*— embedded systems, components, RTOS**

I. Introduction

Embedded operating systems are built to provide specific functionality on specific hardware. Development of a new OS from scratch for every task and hardware is unwise and operating systems are designed to support several CPU architectures and a lot of peripheral devices in a single distribution. Therefore OS distribution contains many drivers to support a large number of different hardware. Most of the drivers are not needed for correct OS execution on a specific board. Moreover many embedded systems are aimed to run in restricted environment, for example with limited memory.

Static OS configuration is used in cases when it is known in advance on which hardware the OS image is going to be executed. OS configuration is commonly performed by the *system integrator*. They choose OS features suitable for OS task and drivers for hardware. Only chosen parts will get into final OS image. System integrator doesn't change OS source code. Static configuration allows keeping final image small.

Safety-critical systems must be certified. For airborne systems there is a standard for certification called DO-178C [1], where OS kernel must be certified by highest level of reliability. Certification is complex and lengthy process. Small change in one part of system leads to recertification of the whole system.

We develop an open-source real-time operating system for civil aircraft airborne computers called JetOS. JetOS is ARINC-653 [2] compliant, supports static configuration and aimed to DO-178 certification.

ARINC-653 specifies interfaces that RTOS (real-time operating system) should provide to avionics software, also the standard specifies some design constrains to the OS. The most pertinent constraint is that application code is executed inside partitions that are isolated from each other by resources and in time.

To simplify and minimize OS kernel and therefore to simplify OS certification process we moved drivers and some services from kernel to special ARINC-653 partitions, called system partitions [3]. Besides drivers system partition contains services such as network stacks, file systems, logging, etc.

System partitions should be certified as well as the kernel. Certification for highly-critical software requires absence of unreachable code. Usage of static configuration of the system partition allows to static selection of required drivers and services, and therefore getting rid of unused code.

It is common that there are many vendors involved in building a specific embedded solution: OS vendor, BSP vendor, device driver developers, system integrator, etc. When services or drivers they are developing are strongly coupled, developers have to interact a lot.

Therefore splitting system partition to independent isolated components seems to be suitable solution. Each driver and service will be in dedicated component. Each component would have a single developer.

Component should interact with each other. Appearance of fixed interface between components would make component development easer. Moreover fixed interface can make system flexible. Statically configured component-based system (in our case system partition) can be flexible in several aspects:

- When there are several components implementing the same interface (e.g. several file systems) and system integrator can choose which component will get into final image.
- When there are several components implementing the same interface, and they all can get into final image. System integrator configure on static, which components interact. For example, if there are two file systems, some component would work with one file system and others with the second one.
- When system integrator can add new component between two interacting, if the new component has a suitable interface. This is useful and can be used, for example, to insert traffic analyzer between protocol stack and network card driver.

Another use-case is to reuse a device driver in an applications stack, such as network card driver in the network stack. Isolated into component the same driver code might serve multiple device instances due to different sets of internal states and configuration parameters. All copies of the component share same driver code, so that each component copy would work with assigned device, would make system scalable and flexible.

Certification of system includes, among others, unit and integration tests. Splitting system partition to components makes certification easier. Component-level tests can be run by component developer. And system integrator doesn't need to rerun unit tests, he only needs to run integration tests.

II. Related works

Classical distributed components models like Enterprise JavaBeans, CORBA, Corba Component Model and DCOM [4,5,6] define components and interfaces between them. Models allow substituting one component with the other one with the same interfaces. Components configuration dynamically configured by brokers. This approach is not suitable for embedded systems with static configuration.

Ideas to separate OS appeared long ago in microkernels. Microkernel architecture's [7,8,9] primary goal is to separates OS into independent servers that could be isolated from each other. Servers interact through inter-process communication (IPC). IPC calls are typed and servers with the same interface can substitute one another. But there cannot be two servers with the same interface; therefore this model is not suitable for our tasks too.

VxWorks is a popular embedded operating system. VxWorks board support package (BSP) is divided into components. Components interface is declared in component description language (CDL). BSP developer can construct BSP from existing component and can add their own components. But this system is not flexible; for example, each component has hardcoded in it a list of names of components it interact with, therefore one component cannot be easily substituted in a configuration with another one with the same interfaces.

We are not aware of any component based model with the following set of features:

- Static configuration,
- Low overhead,
- Flexible configuration (in all aspects from introduction),
- Low mishit probability, when component interact with component it not designed to (runtime addressing checks)

III. Basic capabilites of Component-Based Model

Our model aimed to have small overhead, so it can be suitable for RTOS. In its raw form, our model assumes that there is a lot of similar code written by component developers in C language. To reduce the amount of hand work we generate helper code, based on configuration files. Language, which is used to write configuration files, can be any declarative language; we use YAML for these purposes.

A. *Component developer view*

Model defines component types and component instances. Each component has a unique component type and assigned implementation and any number of instances. Component type is similar to term "class" from object oriented languages and component instance is similar to "class objects". Component instances share code, but sharing does not apply to some private data, called instance state.

Components interact. The ability of one component to use services of the others is achieved through typed ports. There are two kinds of component ports:

- Input ports, which show that the component provides some functionality. Input ports have assigned handlers implemented by the component, which will be called when some other component calls the interface of the component.
- Output ports, which are used by a component when invokes behavior of another component. The component calls others indirectly, through output ports.

Ports are typed, input port of one component and output of the other one can be connected only if they have the same port type. Port type is called interface. Interface is the set of functions, which input port provides or output port require. Since interface can have several functions, then output port that implements this interface has several assigned handlers, one for each function in interface.

Interface declares as the set of triple of function names, signature, and return types. Example of simple interface declaration can be seen at fig.1.

```
- name: data_sender
  functions:
    - name: send
      return_type: ret_t
      args_type: [int]
```

Fig. 1. *Data_sender* interface with one function *ret_t send(int)*

Component type declaration contains component name, component instance state structure, and component ports. Output ports are declared as pair of port name and port interface. Input ports are declared as triple (n, I, m): port name n, port interface I, and m a list of pairs of interface function and assigned implementation specified by components function name.

You can see example of component type configuration at fig. 2.

```
name: Filter
state_struct:
  edge: int

input_ports:
  - name: in
    type: data_sender
    implementation:
     send: filter_send

output_ports:
  - name: out
    type: data_sender
```

Fig. 2. Component type *Filter*. Component state contains one field *edge*. Componet type has single input port called *in*, port interface is *data_sender*, fucntion send of *data_sender* interface is implemented by *filter_send* function.

During system build configuration files are parsed and corresponding C code is generated:

- C-structure describing component, with name identical to component name. (e.g. structure `Filter` for component Filter)
- Declaration of functions specified in input ports (e.g. declaration of function `filter_send` for component Filter). This declaration enforces naming convention.
- Special function for calling output ports.

Component developers should use only ports to communicate with other components. Direct call of another component might work but is not guaranteed. The component developer is guaranteed only the interfaces. The developer chooses names for ports. Input ports is an entry point to component. Names on input ports are not used by component developer. Output ports are used when component should use service of another component. To call the output port a developer should specify output port name, output port function name, and function arguments. Developer should not assume what real function of which component will be called. You can see an example of calling function from output port at fig. 3

```
ret_t filter_send(Filter *self, int data)
{
  ...
  res = Filter_call_out_send(self, data);
  ...
}
```

Fig. 3. Call of function *send* of port *out*.

B. System integrator view

System integrator decides how many instances of each component should be created, and how they are connected. For each component they choose unique name, and how to initialize its state. System integrator uses instance names and names of their ports to link ports of different instances. All of this information integrator specifies in configuration file. Graphical view of example configuration use can see at fig. 4.

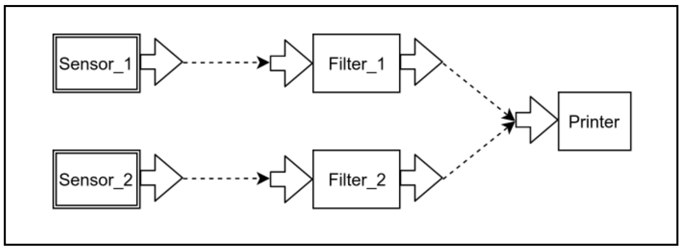


Fig. 4. Example linkage configuration. *Sensor_1* and *Sensor_2* are instances of *Sensor* component type. *Filter_1* and *Filter_2* are instance of *Filter* component type. *Sensor_1* ouput port connect to Filter_1 input port. *Filter_1* input port connected to *Printer*. Same for *Sensor_2* and *Filter_2*

IV. Advanced capabilities of component based model

A. Init function

Instances can have init function: component developers should declare init function name in configuration. At system partition start all init functions of all instances are called sequentially. There is no way to specify dependencies on init (i.e. init of open component should be called before init of the other one) because we assume that components are independent and should not have any dependency.

B. Active and reactive components

All components with input ports are *reactive*, i.e. get control by call from other component. Some components are *active*, i.e. the component gets control from OS by some regularities (periodically or by event). Component can be active and reactive at the same time.

There are two types of active components in our model:

- Components which have a special entry point – activity. This type of active components is useful when component instances should do some simple work from time to time (for example, checking whether there are any new networks packets). Component developer declares activity name in configuration. All activities are called sequentially. This type of active components has a big disadvantage: if some instance will freeze in its activity then all instances of this type in the system are going to freeze, so component developer should not use any wait objects in activity.
- Components, which instances create their own threads inside init function. In this case freezing of the instance, which is running in the dedicated thread, will not cause freezing other instances.

C. Array of ports

Sometimes component developers need to create configurable number of ports of the same type. We support array of ports, but only for output ports. For calling function of output port array developers should specify index in the array besides port name, function name and function arguments.

Arrays of ports are useful in components like router (at the fig. 5). Router sends data to configurable of instances. Integrator in the configuration specifies number of elements in port array and their linkage with instances.

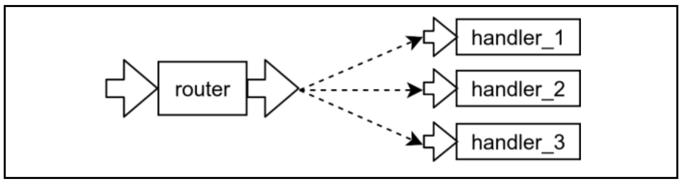


Fig. 5. Router has an array of out port which are connected to instances *handler_1*, *handler_2* and *handler_3*

D. *Memory blocks*

Component instances in our system cannot use system heap, because there can be heap underflow with many instances and not enough heap size.

Access to heap and physical (for drivers) memory is done through ARINC-653 memory blocks. For each memory block component developer specifies:

- memory block name suffix
- memory size
- memory alignment
- flag, that shows if this memory block used by single instance or shared between instances.
- physical address for drivers working with memory mapped devices. Memory blocks with fixed physical address must be shared.

Name of shared memory blocks is identical to name suffix from configuration. Name of non-shared memory block is concatenation of instance name and memory block name suffix. Instances can access memory blocks by ARINC-653 API specifying memory block name.

E. *Memory ownership*

This part of the paper doesn't describe a feature of our approach. Here is some consideration on memory ownership.

Let's consider a component based system partition, which implement networking. There can be a track of components: Message_sender →UDP_IP_sender → Eth_sender→ Network_card_driver. Message sender sends pointer message to UDP_IP_sender; UDP_IP_sender prepends message with UPD and IP header and sends message to Eth_sender; Eth_sender prepends message with Ethernet header and sends to Network_card_driver. Should be specified how own memory and responsible for memory allocations.

If UDP_IP_sender and Eth_sender components would allocate buffers in their own memory, then this would greatly complicate their code, as they should also free buffers. Our real time C library doesn't support memory freeing because memory freeing can make indeterminate amount of time.

To simplify implementation and reduce overhead we used an approach when Message_sender allocates enough memory for all headers (component gets this value from configuration), copies message at the needed offset and pass to next layer pointer to message, message size, prepend and append values. Prepend describes how many bytes before message are allocated. Append describes how many bytes after message are allocated.

UDP_IP_sender to add header moves pointer it gets from Message_sender and decreases prepend value to header size.

V. Future work

We are going to work on supporting component distribution by binary images. This can be used to protect intellectual property of component developer, who doesn't want to share component source code.

Currently system integrator should specify component instances and their linkage in YAML language. We are going to support AADL language, which allows system integrator to graphically create and link instances. To work with AADL we are going to use MASIW framework. MASIW [10,11] (MASIW – Modular Avionics System Integrator Workplace) is s an open source Eclipse-based IDE for development and analysis of AADL models.

Also we are going to research possibility of using dataflow language to specify component, so that there will be no need to write component implementation in C language

VI. Concslusion

In the paper we presented a component based approach that was created for JetOS, but can be used in other systems. The approach turned out to be efficient; it has low overhead and make system flexible and scalable while statically configured.

References

[1] DO-178C, Software Considerations in Airborne Systems and Equipment Certification, January 5, 2012

[2] Avionics application software standard interface part 1 – required services, ARINC specification 653P1-3, November 15, 2010

[3] Mallachiev K.M., Pakulin N.V., Khoroshilov A.V. Design and architecture of real-time operating system. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 2, 2016, pp. 181- 192. DOI: 10.15514/ISPRAS-2016-28(2)-12

[4] J. Siegel, Corba 3 fundamentals and programming, John Wiley & Sons, 2000

[5] Nanbor Wang, Douglas C. Schmidt, and Carlos O'Ryan. 2001. Overview of the CORBA component model. In Component-based software engineering, George T. Heineman and William T. Councill (Eds.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA 557-571.

[6] Distributed Component Object Model (DCOM) Remote Protocol Specification (online)

[7] Alain Gefflaut, Trent Jaeger, Yoonho Park, Jochen Liedtke, Kevin J. Elphinstone, Volkmar Uhlig, Jonathon E. Tidswell, Luke Deller, and Lars Reuther. 2000. The SawMill multiserver approach. In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system (EW 9). ACM, New York, NY, USA, 109-114. DOI=http://dx.doi.org/10.1145/566726.566751

[8] J. Liedtke. 1995. On micro-kernel construction. In Proceedings of the fifteenth ACM symposium on Operating systems principles (SOSP '95), Michael B. Jones (Ed.). ACM, New York, NY, USA, 237-250. DOI: http://dx.doi.org/10.1145/224056.224075

[9] Boule I, Gien M, Guillemont M. CHORUS Distributed Operating Systems, Computing Systems, Vol. I No. 4 Fall 1988

[10] D. V. Buzdalov, S. V. Zelenov, E. V. Kornykhin, A. K. Petrenko, A. V. Strakh, A. A. Ugnenko, and A. V. Khoroshilov. Tools for system design of integrated modular avionics. In Proceedings of the Institute for System Programming of RAS, volume 26, pages 201–230, 2014.

[11] Alexey Khoroshilov, Dmitry Albitskiy, Igor Koverninskiy, Mikhail Olshanskiy, Alexander Petrenko, and Alexander Ugnenko. AADL-based toolset for IMA system design and integration. SAE Int. J. Aerosp., 5:294–299, 10 2012.

Detecting and Tracking Near Duplicates in Software Documentation

Dmitry V. Luciv
Saint-Petersburg State University
Saint-Petersburg, Russia
Email: d.lutsiv@spbu.ru

***Abstract*—Modern software documentation is complicated as software itself. During authoring and maintenance, it gets a lot of "near duplicate" text fragments, which obstruct working with documentation and lower its quality.**

Here we present an algorithm for near duplicate detection which is based on exact software clone detection algorithm, but allows looking for repetitions with variations. This reflects common nature of repetitions in documents with high amount of copy-pasted text.

An algorithm is implemented by the tool which allows to search for near duplicates, track and manipulate them. Using this tool, algorithm evaluation is carried out, including comparison to earlier results.

***Keywords*—software documentation, near duplicates, documentation reuse, adaptive reuse**

I. Introduction

Like software itself, modern software documentation is often large and complicated. Being maintained and authored during software life cycle, it contains a lot of duplicates. After being copied, documentation fragments are corrected in different ways so that different copies of initially similar text fragments become "near duplicates". Text duplicates in software documentation, both exact and near ones, are studied actively [1], [2], [3], [4]. Different kinds of documentation are discussed in [5]. For different documents, duplicates can be either desired or not, but in any case duplicates increase documentation complexity and thus maintenance and authoring cost is increased too.

To simplify documentation maintenance and authoring, an approach was proposed in [6], [7], allowing to search for near duplicates in documents and then apply reuse techniques described in [3], [4], [8].

In this paper an algorithm for near duplicate search is presented. This algorithm generalizes one from [6], [7]. Just as before, it uses Clone Miner [9] software clone detection tool to get an information on exact duplicates (clones), and then combines them. The key difference between two algorithms is that algorithm from [6], [7] only detects near duplicates with single difference (it only combines two exact duplicates with varying text between them), while proposed algorithm detects near duplicates with arbitrary number of varying text fragments (it combines many exact duplicates at once). To speedup proposed algorithm, an *interval tree* [10] is used. Presented algorithm is implemented in Documentation Refactoring Toolkit [11], which is part of DocLine project [3]. Documentation Refactoring Toolkit allows searching for near duplicate text in documentation and executing a subset of refactoring operations described in [12] basing on search results. In this paper, an evaluation of proposed algorithm is also presented, using documentation of four open source projects as objects of evaluation: Linux Kernel [13], Zend Framework [14], DocBook [15], Subversion [16].

II. Background

A. Exact duplicate detection and Clone Miner

Not only documentation, but also software itself is often developed with a lot of copy-pasted information. To coup with duplicates in source code, software clone detection methods and tools are used. This area is quite mature; systematic review of clone detection methods and tools can be found in [17].

In this paper, Clone Miner [9] software clone detection tool is used to detect exact duplicates in documentation. Clone Miner is a token-based source code clone detector. It considers an input text as a collection of lexical tokens and applies enhanced suffix array based string matching algorithms [18] to find the repeated parts (clones) of this text as clone groups.

Clone Miner is chosen here for its simplicity and ability to be easily integrated with other tools using command line interface.

B. Interval Tree

Interval tree [10] is the data structure, allowing to speed up searching the set of intervals for intersecting ones. Each node of interval tree may be either empty (ε) or not. Node is defined as $T = \varepsilon$ or $\langle c, C, L, R \rangle$, where c is center point, C is a collection of intervals, L and R are references to left and right subtrees, which are also interval trees. Non-empty nodes satisfy following conditions:

1) $\forall i \in C,\ c \in i$
2) $\forall l = [a, b] \in L,\ b < c$
3) $\forall r = [a, b] \in R,\ a > c$

Interval tree implementations balance trees to keep numbers of intervals in left and right subtrees of each non-empty node as close to each other, as possible. The same condition is met for heights of left and right subtrees of each non-empty node.

When expanding intervals on both ends before using them to construct a tree, resulting tree also not only allows searching for intersecting intervals, but also for nearby ones. This approach is used in proposed algorithm.

Documentation Refactoring Toolkit is developed in Python and uses “intervaltree” Python library [19] as interval tree implementation.

III. Near duplicate detection algorithm

A. Definitions

Let us define some terms necessary for describing proposed algorithm below.

Exact duplicate group G_i represents a set of similar text fragments. Those fragments are denoted as *exact duplicates*, which are intervals in document text: $g_i^k = [a_i^k, b_i^k] \in G_i$, where a_i^k and b_i^k are offsets from beginning of the document given in symbols.

In this paper we consider near duplicate groups which consist of near duplicates — fragments of texts, having much in common. Near duplicates can be defined and detected in different ways. Here near duplicate groups are combined from collections of exact duplicate groups, when they meet additional conditions, which are described below.

Having exact duplicate groups $G_1, \dots G_N$, we say that they form *variational group* $VG = \langle G_1, \dots G_N \rangle$, when following conditions are satisfied:

1) $G_1, \dots G_N$ groups have the same number of exact duplicates: $\#G_1 = \dots = \#G_N$.
2) Corresponding exact duplicates of all participating groups do not intersect: $\forall n, m \in [1, N]$, $\forall k \in [1, \#G_1]$ $g_n^k \cap g_m^k = \emptyset$.
3) Exact duplicates of all groups appear in the same order in the text: $\forall n, m \in [1, N]$, $\forall k_1, k_2 \in [1, \#G_1]$ $n < m \implies Before(g_n^{k_2}, g_m^{k_2})$, where $Before([a_1, b_1], [a_2, b_2])$ means that $b_1 < a_2$.

For genericity and simplicity, here and below, when needed, we can implicitly consider exact duplicate groups G_i as variational ones, formed by those exact duplicate groups: $VG = \langle G_i \rangle$.

When we choose a exact duplicate or variational group, it should be possible to compare a distances from this group to other groups and select the closest one to form new variational group of them two. Let us define the distance between exact duplicates $g_1^k \in G_1$ and $g_2^k \in G_2$ as follows. When $g_1^k \cap g_2^k \neq \emptyset$ (they intersect), we assume $dist(g_1^k, g_2^k) = 0$. Otherwise, when $Before(g_1^k, g_2^k)$ and $g_1^k = [a_1, b_1]$, $g_2^k = [a_2, b_2]$, distance is computed as $dist(g_1^k, g_2^k) = a_2 - b_1$.

The distance between G_1 and G_2 is defined as $dist(G_1, G_2) = \max_{k \in [1, \#G]}(dist(g_1^k, g_2^k))$. For both exact duplicate and variational groups, we define distance as $dist(VG_1, VG_2) = \max_{G_1 \in VG_1, G_2 \in VG_2} dist(G_1, G_2)$.

For each exact duplicate or variational group we define its $length$ as a number of symbols covered by its duplicates: $length(\langle G_1, \dots G_N \rangle) = \sum_{n=1}^{N} length(G_n)$, and $length(G_i) = \sum_{k=1}^{\#G_i} |g_i^k|$, where $g_i^k = [a_i^k, b_i^k]$ and thus $|g_i^k| = |[a_i^k, b_i^k]| = b_i^k - a_i^k$.

We define *near duplicate* group as variational group $\langle G_1, \dots G_N \rangle$ in which the distances between exact duplicate groups satisfy the following conditions:

1) For each $n \in [1, N-1]$, $\forall k, dist(g_n^k, g_{n+1}^k) \leq length(g_n^k) + length(g_{n+1}^k)$.
2) For each $n \in [1, N-1]$, for $\{dist(g_n^k, g_{n+1}^k) | k \in [1, \#G_1]\}$, *coefficient of variation* ν does not exceed 40%, where $\nu = \frac{\sigma}{\mu}$, σ^2 is variance and μ is mean [20].

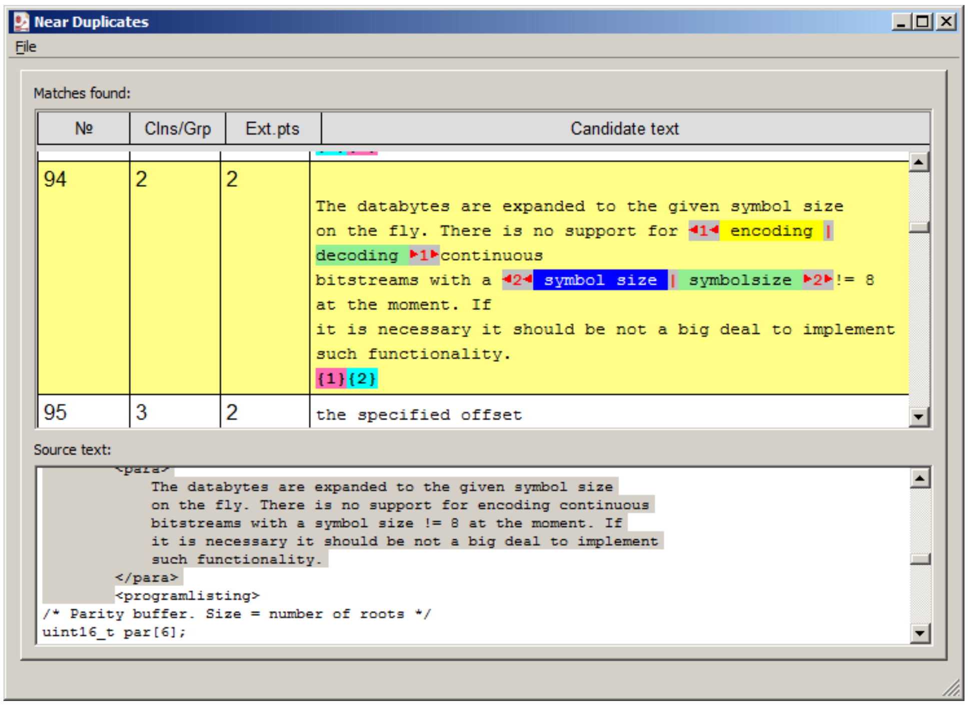
First condition limits variational part size (in symbols) to the whole near duplicate size ratio for any near duplicate of the group. It is easy to prove that for any N, according to this condition, total size of variational parts will not exceed $\frac{2}{3}$ of the whole near duplicate size. This condition is inspired by analogous adaptive reuse condition from [21], [22], but here it is weakened for algorithm clarity and simplicity.

Second condition requires that variation of distances (in symbols) between exact duplicates of G_n and G_{n+1} for $\forall k$ is not too high. For example, when $dist(g_n^1, g_{n+1}^1) = 1$, and $dist(g_n^2, g_{n+1}^2) = 10000$, we will get $\nu \approx 141\%$, showing that there is likely no chance for these pairs to be semantically connected, and it would not be sensible to consider variational group $\langle G_1, \dots G_N \rangle$ as near duplicate group. The threshold value of 40% was chosen as the most suitable during our experiments.

For near duplicate group $\langle G_1, G_2 \rangle$, all of its near duplicates are represented by text of $g_1^k = [a_1^k, b_1^k]$, text of $g_2^k = [a_2^k, b_2^k]$ and some variable text between them, which depends on k and which is placed in $[b_1^k + 1, a_2^k - 1]$. We say, that $\langle G_1, G_2 \rangle$ has single *extension point*, and values of above variable text are *extension point values*. In general case, near duplicate group $\langle G_1, \dots G_N \rangle$ has $N - 1$ *extension points*, and each of them has $\#G_1$ values.

B. Algorithm description

Let us overview an algorithm that constructs near duplicate groups from exact duplicate groups found by Clone Miner. First, we construct an interval tree from exact duplicates, that allows us, having an exact duplicate of some group, to find all exact duplicates (and their groups) which are located close to given one **(step 1)**. After that, we enumerate all groups and their exact duplicates, and, with help of interval tree, find all corresponding groups such that we can combine pairs of groups them into variational groups. Looking at distances' values and variations, we select best pairs possible, producing new near duplicate groups and leaving some groups alone **(step 2)**. Then we update references in interval tree so that they point to newly created near duplicate groups where needed **(step 3)**. Next we try to combine near duplicate groups with both near and remaining exact duplicate groups to expand and join near duplicate groups where available, and also update references in interval tree to keep it up to date **(step 4)**. We try to combine/join near duplicate groups until, after some iteration, we fail to change anything. Then we say that everything possible is done and stop **(step 5)**. As a result, we have the sets of newly constructed near duplicate and remaining exact duplicate groups.



№	Clns/Grp	Ext.pts	Candidate text
94	2	2	The databytes are expanded to the given symbol size on the fly. There is no support for ◂1◂ encoding | decoding ▸1▸continuous bitstreams with a ◂2◂ symbol size | symbolsize ▸2▸!= 8 at the moment. If it is necessary it should be not a big deal to implement such functionality. {1}{2}
95	3	2	the specified offset

```
        <para>
            The databytes are expanded to the given symbol size
            on the fly. There is no support for encoding continuous
            bitstreams with a symbol size != 8 at the moment. If
            it is necessary it should be not a big deal to implement
            such functionality.
        </para>
        <programlisting>
/* Parity buffer. Size = number of roots */
uint16_t par[6];
```

Figure 1. Near duplicate browser of Documentation Refactoring Toolkit

An algorithm is described below in details. It takes exact duplicate group set, denoted as $SetG$, from Clone Miner as input. It removes some groups from $SetG$ to produce near duplicate groups from them, and returns newly produced near duplicate groups as $SetVG$.

Step 1. All exact duplicates are used to construct an interval tree, such that keys in this tree are intervals in document text, covered by exact duplicates, and values are groups, to which those duplicates belong. Interval tree is used here to search for nearby intervals, not intersecting ones. Following first condition of near duplicate definition, “nearby” means that distance between two exact duplicates should not be greater than sum of their lengths. To detect nearby intervals with interval tree, we expand all intervals on their lower and upper boundaries before using them as tree keys, allowing nearby intervals to become intersecting ones.

Intervals are expanded by duplicate length at both the beginning and end: for duplicate $g_n^k = [b, e]$, we use $[b - |[b, e]|, e + |[b, e]|]$ interval as tree key. Let us make sure that suggested expansion is enough. Imagine that we have exact duplicates $g_1^k = [b_1, e_1]$, $g_2^k = [b_2, e_2]$, $Before(g_1^k, g_2^k)$, and they are placed nearby each other: $b_2 - e_1 \leq |[b_1, e_1]| + |[b_2, e_2]|$. Then, after expanding them to $[b_i', e_i'] = [b_i - |[b_i, e_i]|, e_i + |[b_i, e_i]|]$, we can easily prove that $e_1' \geq b_2'$, which guarantees $[b_1', e_1'] \cap [b_2', e_2'] \neq \emptyset$. Therefore, using expanded $[b'^k_i, e'^k_i]$ intervals to construct interval tree as shown above, allows us to detect initial nearby intervals $[b_i^k, e_i^k]$ with it.

Step 2. For each $G_i \in SetG$ we do following:

1) We consider all its exact duplicates g_i^k and, using interval tree, find all exact duplicate groups $\{G_j\}$, having their duplicates g_j^k placed nearby g_i^k.
2) We do not consider all G_j, which do not correspond to G_i in sense of forming near duplicate by checking near duplicate group conditions defined in (III-A).
3) Then we find closest group $G_m \in \{G_j\}$: $m = \operatorname{argmin}_j dist(G_i, G_j)$.
4) After all, we remove both G_i and G_m from $SetG$ and add $\langle G_i, G_m \rangle$ (or $\langle G_m, G_i \rangle$, depending on their duplicates placement) to $SetVG$.

Due to interval tree, for N total exact duplicates in all groups, when any of them can be combined with at most M another groups, this step takes $\mathcal{O}(N(\log N + M))$ time in average. In [6], [7] this was the only step of algorithm and, having no interval tree, it took $\mathcal{O}(N^2)$ time to complete.

Step 3. We update references to groups in interval tree so they point to newly created near duplicate groups for expanded intervals of their exact duplicates.

Step 4. Then we try to expand and join near duplicate groups their selves. For each near duplicate group in $SetVG$, we perform the same procedure as described in steps 2 and 3, except:

1) We take G_j from $SetG \cup SetVG$.
2) We add (prepend or append) new exact and near duplicate groups to existing near duplicate ones instead of constructing new near duplicate groups. Then obsolete exact and near duplicate groups are removed from $SetG$ and $SetVG$ respectively.

Step 5. We repeat Step 4, until at some iteration no changes happen to $SetG$. Then $SetVG$ construction is finished. Fi-

Table I
EVALUATION RESULTS

Documentation	Source exact dup. grps	Algorithm from [6],[7]			Proposed algorithm				
		Time, s.	Near dup. grps found	Exact dup. grps involved	Time, s.	Near dup. grps found	Exact dup. grps involved	Average ext pts.	Max ext pts.
LKD	13881	705	242	484	408	308	682	1.21	5
Zend	36841	4981	1035	2070	2824	1067	2550	1.39	10
DocBook	8951	293	174	348	137	210	480	1.28	19
SVN	29603	3184	442	884	961	471	1085	1.28	10

nally, algorithm results in modified $SetG$ and newly built $SetVG$ as its results.

IV. THE TOOL

Above algorithm is implemented by Documentation Refactoring Toolkit [11]. This tool can operate as both standalone one and an integrated part of DocLine environment.

After being given input document, it produces the report as shown on Fig. 1. Report contains both exact and near duplicates, detected in the document. When shown to user, they are sorted by their lengths decreasing. User can browse duplicates found. When (s)he clicks on extension point value or duplicate number, document source pane at the bottom selects corresponding text fragment. Generated report can be exported to standalone HTML file using *File* menu. In addition to detection and browsing of near duplicates, Documentation Refactoring Toolkit also provides refactoring facilities, which are already described in [6], [7] and which are not affected by proposed algorithm.

V. EVALUATION

Let us characterize the way of assessment in out experiments with two questions:

- **Question 1:** How many near duplicate groups with single and multiple extension points can be found in real life documents?
- **Question 2:** How do results and performance differ from results and performance of algorithm described in [6], [7]?

To assess above questions, the both proposed and described in [6], [7] algorithm implementations were evaluated against four open source projects' documentation:

- Linux Kernel documentation (acronym: LKD), 892 KB in total [13];
- Zend Framework documentation (acronym: Zend), 2924 KB in total [14];
- DocBook 4 Definitive Guide (acronym: DocBook), 686 KB in total [15];
- Version Control with Subversion (acroym: SVN), 1810 KB in total [16].

Evaluation results for each document are presented in Table I. *Documentation* column contains evaluation objects' acronyms. *Source exact dup. grps* shows us, how many exact duplicate groups were found by Clone Miner for particular documents. The rest presents results for algorithm from [6], [7] and proposed one. For both algorithms, column *Time, s.* shows time, in seconds, spent to detect near duplicates (the same computer was used for all experiments); *Near dup. grps found* column shows, how many near duplicate groups were detected; *Exact dup. grps involved* column contains number of exact duplicate groups involved in near duplicate groups construction. Then, only for proposed algorithm, *Average ext pts.* column shows, how many extension points do resulting near duplicates have in average; the last *Max ext pts.* column presents maximal number of extension points found.

To answer **question 1**, we can compare numbers of exact duplicate groups found by Clone Miner and numbers of near duplicate groups found by proposed algorithm. The most of exact duplicate groups remain so, but there are also considerable amount of near duplicate groups built from them, having 1.2 to 1.4 extension points in average for different documents.

To answer **question 2**, we can compare total counts of near duplicate groups produced by algorithm from [6], [7] and proposed algorithm. Proposed one constructs 9% more near duplicate groups and involves 27% more exact duplicate groups for it. Thus we can say that it joins exact duplicate groups and constructs near duplicate ones more effectively. When comparing time spent to process real documents, proposed algorithm shows itself being 2.24 times faster in average, having maximum speedup of 3.3 times over one from [6], [7].

VI. CONCLUSION

Following evaluation results, we see that proposed algorithm offers notifiable improvements over algorithm described in [6], [7]. The main improvement is that it detects near duplicates with arbitrary number of extension points. It is essential, because real life documents contain considerable amounts of such near duplicates.

However we should also mention that proposed algorithm has some limitations. First, it only searches for near duplicates with extension points placed between exact duplicates. It does not perform any semantic analysis of processed text, thus having only first and last exact duplicates as boundaries for constructed near duplicate. Next, in [6], [7], near duplicates followed the rule, saying that average size of variational part of near duplicate should be at most $\frac{1}{2}$ of total near duplicate size. The idea of limiting variational part size to the total near duplicate size ratio comes from [21], [22]. Proposed algorithm also forces $\frac{1}{2}$ constraint for near duplicates with single extension point, but when having multiple extension points, variational part size can reach up to $\frac{2}{3}$ of total near duplicate size.

After looking through particular near duplicates in evaluation results, we can say that first limitation appears to be more significant. Following this, in the future we plan to extend proposed approach with some heuristic text analysis and natural language processing, thus improving near duplicate boundaries detection.

Acknowledgment

This work is partially supported by RFBR grant No 16-01-00304.

References

[1] E. Juergens et al. Can clone detection support quality assessments of requirements specifications? In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering*, vol. 2., 2010, pp. 79–88

[2] M. Nosáľ, J. Porubän. Preliminary report on empirical study of repeated fragments in internal documentation. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, 2016.

[3] D. Koznov, K. Romanovsky. DocLine: A Method for Software Product Lines Documentation Development. In *Programming and Computer Software*, 34(4), 2008, pp. 216–224

[4] K. Romanovsky, D. Koznov, L. Minchin. Refactoring the Documentation of Software Product Lines. CEE-SET 2008, Brno (Czech Republic), October 13–15, 2008. In *LNCS*, vol. 4980, Springer 2011, pp. 158–170

[5] D. L. Parnas. Precise Documentation: The Key To Better Software. In S. Nanz, editor, *The Future of Software Engineering*, Springer, 2011

[6] D. Koznov et al. Clone detection in reuse of software technical documentation. In: Mazzara, M., Voronkov, A. editors, *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2015, *Lecture Notes in Computer Science*, vol. 9609, 2016, pp. 170–185

[7] D. Luciv, D. Koznov, H. A. Basit, A. N. Terekhov. On fuzzy repetitions detection in documentation reuse. In *Programming and Computer Software* 42(4), 2016, pp. 216–224

[8] K. Romanovsky, D. Koznov. DRL: a Language for Software Product Line Documentation Design and Development In *Vestnik Of Saint Petersburg University, series 10: Applied Mathematics. Computer Science. Control Processes.* Issue 4, pp. 110–122, 2007 (in Russian)

[9] H. A. Basit et al. Efficient Token Based Clone Detection with Flexible Tokenization. In *Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2007, pp. 513–516

[10] Mark de Berg et al. Computational Geometry, Second Revised Edition. Springer-Verlag 2000. Section 10.1: Interval Trees, pp. 212–217.

[11] Document Refactoring Toolkit, http://www.math.spbu.ru/user/kromanovsky/docline/index_en.html

[12] D. Koznov, K. Romanovsky. Automated Refactoring of Software Product Line Documentation In *System Programming*, vol. 4, pp. 128–150, 2009 (in Russian)

[13] Linux Kernel Documentation, snapshot on Dec 11, 2013, https://github.com/torvalds/linux/tree/master/Documentation/DocBook/

[14] Zend PHP Framework documentation, snapshot on Apr 24, 2015, https://github.com/zendframework/zf1/tree/master/documentation

[15] DocBook Definitive Guide, snapshot on Apr 24, 2015, http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/

[16] SVN Book, snapshot on Apr 24, 2015, http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/

[17] D. Rattan, R. K. Bhatia, M. Singh: Software Clone Detection: A Systematic Review. Information & Software Technology (INFSOF), 55(7), pp. 1165–1199 (2013)

[18] M. I. Abouelhoda, S. Kurtz, E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. In *Journal of Discrete Algorithms*, 2(1), 2004, pp. 53–86

[19] A mutable, self-balancing interval tree for Python 2 and 3. https://github.com/chaimleib/intervaltree

[20] B. Everitt. The Cambridge Dictionary of Statistics. Cambridge, UK New York: Cambridge University Press, 1998

[21] P. G. Bassett. Framing Software Reuse: Lessons from the Real World. *Prentice-Hall, Inc., Upper Saddle River, NJ, USA*, 1996

[22] P. G. Bassett. The Theory and Practice of Adaptive Reuse. In *SIGSOFT Software Engineering Notes*, 22(3), 1997, pp. 2–9

Discovering Near Duplicate Text in Software Documentation

Leonid D. Kanteev
Saint-Petersburg State University
Saint-Petersburg, Russia
Email: lkolt2@mail.ru

Yuri O. Kostyukov
Saint-Petersburg State University
Saint-Petersburg, Russia
Email: taxixx@inbox.ru

Dmitry V. Luciv
Saint-Petersburg State University
Saint-Petersburg, Russia
Email: d.lutsiv@spbu.ru

***Abstract*—Authoring and maintaining software documentation often involves copy-pasting text and editing it afterwards, which produces a lot of near duplicate text. Such duplicates make it difficult and expensive to work with documentation, especially because modern software documentation is often complicated and has long lifecycle, as software itself. Usually, there are no special efforts to track those duplicates.**

In this article, we present an algorithm, aiming to detect near duplicates in software documentation. An algorithm relies on natural language processing techniques, its implementation uses Natural Language Toolkit software library for text analysis.

Algorithm implementation is evaluated, using documentation of several open source projects as evaluation objects.

***Keywords*—software documentation, near duplicates, natural language processing**

I. Introduction

Todays technical writers, software engineers, architects and managers often deal with large and complicated software documentation. When being authored or maintained, this documentation gets a lot of copy-pasted text, which is often not tracked properly. In general, the topic of documentation reuse in software engineering is studied actively [1], [2], [3], [4].

According to software documentation classification from [5], there are different kinds of documents. For some of them, duplicate text is undesired, while other ones should contain exact or near duplicate text, but in the case of near duplicates in the document, those near duplicates should still remain as uniform as possible. Basing on text fragment reuse, Nosál et al. [6] and Horie et al. [7] offer to solve the problem of API documentation unification, Romanovsky et al. [4] suggest refactoring XML-documents. But the problem of effective duplicates search is still open, because we need to get not exact, but near duplicates [8]. There are some case studies [6], [9] and approaches [1], [8], [10], [11], but all of them apply clone detection technique to documentation duplicate search. Koznov et al. [8], [10] use exact duplicates which are found by means of clone detection to organize near duplicate search. But this approach leads to processing a lot of false positive information [9], and it is hard to provide proper accuracy of the final results.

In this paper we suggest finding near duplicates in software documentation without clone detection technique and omitting intermediate phases of exact duplicate search. Doing that we aim search accuracy increasing by providing filtering of false positive data on the "low level". We use Natural Language Toolkit [12] (NLTK), and present an algorithm for searching near duplicates, which consist of one sentence. We have implemented this algorithm and evaluated the results on evaluation objects used earlier in [8], [10].

II. Background

Modern natural language processing and computer linguistics employ a lot of different standard algorithms and approaches to transform, normalize and analyze texts. Proposed approach and algorithm require raw input document to be preprocessed: text should be divided into sentences, sentences should be divided into words, words should be grouped to n-*grams*. Here n-grams, also known as *shingles*, are overlapping sequences of n words, starting from each word of the text [13].

Sometimes, those operations are more complicated than they appear at first glance, because of language-specific idioms and abbreviations. For such text manipulations, Natural Language Toolkit [12] (NLTK) is popular choice. NLTK is a collection of software libraries for natural language processing and text analysis. It is suitable for our needs because it already contains implementations of many standard linguistic operations, including those described above, for many natural languages and also because it is implemented in Python, making itself easy to integrate into Documentation Refactoring Toolkit [14].

III. An algorithm for sentence grouping

Here we propose an algorithm that groups document sentences, which contain near duplicate text and were likely derived from similar text fragments.

First, we execute preliminary text processing with NLTK to extract sentences and 3-grams (n-grams of three words) from it. All sentences are represented by their bindings to source document and sets of 3-grams, extracted from their texts. Sentence group is represented by its sentences and set of all 3-grams of those sentences. For each sentence, algorithm scans existing groups and selects best one, which aready contains the largest number of sentence 3-grams. Then, if the best group already contains at least one half of sentence 3-grams, sentence is added to this group, and group's 3-grams set is complemented with sentence's 3-grams. When no groups containing half of sentence 3-grams are found, new group, containing all sentence's 3-grams, is introduced. Last, algorithm outputs sentences of all groups, which contain two or more entences, as near duplicates.

Let us present **Algorithm 1** pseudocode. We use following conventions there:

- *intersect(A, B)* function returns element, which exist in the both A and B arrays
- *size(A)* function returns number of elements in array A
- *sent* is an array of sentences in document text
  - *sent[i].nGrams* is an array of 3-grams of i-th sentence
- *groups* is an array of near duplicate groups
  - *groups[i].nGrams* is an array of 3-grams of i-th group
  - *groups[i].sent* is an array of sentences of i-th group

Algorithm 1 Near duiplicate group detection

```
1:  for i = 1 to size(sent) do
2:    curSent = sent[i]
3:    bestOverlap = 0
4:    bestGroup = NULL
5:    for j = 1 to size(groups) do
6:      curGroup = groups[j]
7:      curIntersect =
          ↪ intersect(curSent.nGrams, curGroup.nGrams)
8:      curOverlap = size(curIntersect) / size(curSent.nGrams)
9:      if curOverlap > bestOverlap then
10:       bestOverlap = curOverlap
11:       bestGroup = curGroup
12:     end if
13:   end for
14:   if bestOverlap < 0.5 then
15:     create new group newGroup
16:     newGroup.nGrams += curSent.nGrams
17:     newGroup.sent += curSent
18:   else
19:     bestGroup.nGrams += curSent.nGrams
20:     bestGroup.sent += curSent
21:   end if
22: end for
23: for all G in groups such that size(G) = 1, groups -= G
24: return groups
```

Details of proposed algorithm are described below:

1) **Lines 1–22:** main cycle, which iterates over all sentences of the document.
2) **Lines 5–13:** the cycle of the best group selection. For each of existing groups:
   a) **Line 7:** we calculate the intersection of its 3-grams set with the set of current sentence 3-grams.
   b) **Line 8:** we calculate this intersection size to total sentence 3-grams set size ratio.
   c) **Lines 9–12:** if current group is the best of processed ones, we remember it.
3) **Line 14:** we check if above ratio is less than 0.5, and:
   a) **Lines 15–17:** when it is less than 0.5, we create new group and put sentence into it.
   b) **Lines 19, 20:** otherwise, we put the sentence into the best group found.
4) **Line 23:** groups with single sentence are not near duplicate groups, therefore we remove them.
5) **Line 24:** we return all sentence groups, containing two or more sentences.

IV. Evaluation

Following GQM approach [15], we characterize the way of the assessment in our experiments with two questions:

- **Question 1:** How many incorrect, irrelevant and meaningful near duplicates are found?
- **Question 2:** What *reuse amount* do those meaningful duplicates provide?
- **Question 3:** How are algorithm evaluation results compared to manual document analysis results?

We use the term *reuse amount* from [16], which means total number of symbols, covered by duplicates to total document length in symbols ratio. In [1] the same metric is named *clone coverage*.

To assess above questions, algorithm implementation was evaluated against four open source and one commercial projects' documentation:

- Linux Kernel dokumentation (acronym: LKD), 892 KB in total [17];
- Zend Framework dokumentation (acronym: Zend), 2924 KB in total [18];
- DocBook 4 Definitive Guide (acronym: DocBook), 686 KB in total [19];
- Version Control with Subversion (acroym: SVN), 1810 KB in total [20];
- Commercial project user guide (acronym: CProj), 164 KB in total.

Results are presented in table I, which is organized as follows: *Acronym* column contains evaluation objects' acronyms listed above. Then the group of five columns (*Proposed algorithm*) represent results on near duplicate groups found by proposed algorithm: *Total* column shows total numbers of groups found; *Markup-only* column contains numbers of groups without human-readable text (they only contain markup); *Irrelevant* column contains numbers of false-positive groups, which were detected by human during manual revision; *Total meaningful* column shows number of meaningful duplicates; *Meaningful reuse amount* column presents reuse amount, provided by those duplicates. Last column group (*Manual analysis*) contains results of manual text analysis: *Total meaningful* column shows total numbers of meaningful near-duplicate groups found by human; *Meaningful reuse amount* column contains reuse amount, provided by above near duplicates.

To answer **question 1**, we compare numbers in *Total*, *Markup-only*, *Irrelevant* and *Total meaningful* columns of table I for proposed algorithm. 14.4% of groups contain no

Table I
NEAR-DUPLICATE GROUPS DETECTED

Acronym	Proposed algorithm					Manual analysis	
	Total	Markup-only	Irrelevant	Total meaningful	Meaningful reuse amount	Total meaningful	Meaningful reuse amount
LKD	189	38	25	126	7.7%	15	5.1%
Zend	601	62	159	380	8.6%	27	2.1%
DocBook	73	10	24	39	3.2%	12	1.7%
SVN	349	97	75	177	5.0%	16	2.3%
CProj	72	0	21	51	29.5%	9	14.1%
Average reuse amount					10.8%		5.0%

human-readable text, but only markup, 24.6% of groups contain text which is similar enough, but this formal similarity is irrelevant, and duplicates of those groups are not semantically connected. Remaining 61% of groups are meaningful duplicate groups. For documents of different sizes their count varies from few dozens to several hundreds depending on the size and nature of document, therefore we can say that proposed algorithm detects considerable amount of near duplicates, and most of them are meaningful.

Assessing **question 2**, we calculate average reuse amount (table I, *Meaningfull reuse amount* column for proposed algorithm) provided by above meaningful duplicates and see that it is 10.8%.

Last, we answer **question 3** by comparing meaningful near duplicate groups' numbers and meaningful reuse amounts, given by proposed algorithm and obtained from manual analysis results. Information is presented in *Total meaningful* and *Meaningful reuse amount* columns of table I. Proposed algorithm detects 8.5 times more groups in average for all documents, and total provided reuse amount is 2.14 times higher than for manual analysis. Large number of near-duplicate groups detected is caused by algorithm limitation: it only searches for single sentence near duplicates, thus detecting many separate groups of small text fragments. Additional review of resulting near duplicate groups shows that significant difference in reuse amounts was caused by different people making their consequences on near duplicate groups' meaningfulness, thus using different subjective criteria for it.

V. CONCLUSION

Here we presented and evaluated an algorithm for detection of near duplicates in software documentation. Proposed algorithm is close to naive voting clustering algorithm [21], using similarity measure resembling Jaccard index [22]. Compared to [8], [10] it is much simpler, and it also makes use of those techniques and apparatus, which are conventionally used for text analysis.

At present, our algorithm has an important limitation: it only suggests single sentences as near duplicates. Our primary goal for future research is overcoming this limitation to handle not only single sentence near duplicates, but also arbitrary self-sufficient text fragments.

Continuing our research, we can also name other important future directions for it:

1) *Improvement of experiment design, aided to eliminate or estimate subjectivity as possible.* Juergens et al. applied a lot of effort to obtain objective results [1]. Significant difference in reuse amounts (**question 3**, section IV, Evaluation) shows that we definitely should pay more attention to experiment accuracy and reliability in our future research.
2) *Additional semantical classification of near duplicate groups.* Meaningful near duplicates usually describe entities of the same nature (function descriptions, command line parameters, data type specifications, etc.) Knowing the nature of particular duplicate groups, we can perform more detailed analysis of their duplicates and draw more conclusions from it.
3) *Explicit retrieval of common text parts of near duplicates.* Not only it is interesting to detect near duplicate text groups in the document, but also to reestablish their structures from similar and different parts of their near duplicates. Such a facility should provide an assistance in the both document understanding and proper reuse organization [23], which can be done by implementing refactoring operations described in [4], [24].

ACKNOWLEDGMENT

This work is partially supported by RFBR grant No 16-01-00304.

REFERENCES

[1] E. Juergens et al. Can clone detection support quality assessments of requirements specifications? In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering*, vol. 2., 2010, pp. 79–88

[2] M. Nosál, J. Porubän. Preliminary report on empirical study of repeated fragments in internal documentation. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems*, 2016.

[3] D. Koznov, K. Romanovsky. DocLine: A Method for Software Product Lines Documentation Development. In *Programming and Computer Software*, 34(4), 2008, pp. 216–224

[4] K. Romanovsky, D. Koznov, L. Minchin. Refactoring the Documentation of Software Product Lines. CEE-SET 2008, Brno (Czech Republic), October 13–15, 2008. In *LNCS*, vol. 4980, Springer 2011, pp. 158–170

[5] D. L. Parnas. Precise Documentation: The Key To Better Software. In S. Nanz, editor, *The Future of Software Engineering*, Springer, 2011

[6] M. Nosál, J. Porubän. Reusable software documentation with phrase annotations. *Central European Journal of Computer Science*. 4(4), 2014, pp. 242–258
[7] Horie, M., Chiba, S. Tool support for crosscutting concerns of API documentation. In *Proceedings of 9th International Conference on Aspect-Oriented Software Development*, 2010, pp. 97–108
[8] D. Koznov et al. Clone detection in reuse of software technical documentation. In: Mazzara, M., Voronkov, A. editors, *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2015, *Lecture Notes in Computer Science*, vol. 9609, 2016, pp. 170–185
[9] E. Juergens, et al. Can clone detection support quality assessments of requirements specifications? April 2010 ICSE '10: In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, vol 2, pp. 79–88
[10] D. Luciv, D. Koznov, H. A. Basit, A. N. Terekhov. On fuzzy repetitions detection in documentation reuse. In *Programming and Computer Software* 42(4), 2016, pp. 216–224
[11] D. Lutsiv et al. An approach for clone detection in documentation reuse. In *Scientific and Technical Journal of Information Technologies, Mechanics and Optics* 4(92), 2014, pp. 106–114 (in Russian)
[12] Natural Language Toolkit, http://nltk.org/
[13] A.Z. Broder et al. Syntactic clustering of the web. *Computer Networks and ISDN Systems*. 29 (8), 1997, pp. 1157–1166
[14] Document Refactoring Toolkit, http://www.math.spbu.ru/user/kromanovsky/docline/index_en.html
[15] V. Basili, G. Caldiera, H. Rombach. The Goal Question Metric Approach. *Encyclopedia of Software Engineering*, Wiley, 1994
[16] W. Frakes, C. Terry. Software reuse: metrics and models. In *ACM Comput. Surv*, 28(2), 1996, pp. 415–435
[17] Linux Kernel Documentation, snapshot on Dec 11, 2013, https://github.com/torvalds/linux/tree/master/Documentation/DocBook/
[18] Zend PHP Framework documentation, snapshot on Apr 24, 2015, https://github.com/zendframework/zf1/tree/master/documentation
[19] DocBook Definitive Guide, snapshot on Apr 24, 2015, http://sourceforge.net/p/docbook/code/HEAD/tree/trunk/defguide/en/
[20] SVN Book, snapshot on Apr 24, 2015, http://sourceforge.net/p/svnbook/source/HEAD/tree/trunk/en/book/
[21] Ronald K. Braun and Ryan Kaneshiro. 2003. Exploiting topic pragmatics for new event detection. Technical report. In *National Institute of Standards and Technology, Topic Detection and Tracking Workshop*, 2004
[22] P. Jaccard. Distribution de la flore alpine dans le Bassin des Dranses et dans quelques regions voisines. In *Bull. Soc. Vaudoise sci. Natur* 37(140), 1901, pp. 241–272 (in French)
[23] K. Romanovsky, D. Koznov. DRL: a Language for Software Product Line Documentation Design and Development In *Vestnik Of Saint Petersburg University, series 10: Applied Mathematics. Computer Science. Control Processes.* Issue 4, pp. 110–122, 2007 (in Russian)
[24] D. Koznov, K. Romanovsky. Automated Refactoring of Software Product Line Documentation In *System Programming*, vol. 4, pp. 128–150, 2009 (in Russian)

The Program for Public Mood Monitoring through Twitter Content in Russian

Sergey Smetanin
Faculty of Business and Management
National Research University Higher School of Economics
Moscow, Russia
sismetanin@gmail.com

Abstract — **With the popularization of social media, a vast amount of textual content with additional geo-located and time-stamped information is directly generated by human every day. This paper aims at describing the development of the program for public mood monitoring based on sentiment analysis of Twitter content in Russian. Machine learning and natural language processing techniques were used for the program implementation. As a result, the client-server program was implemented, where the server-side application collects and analyses tweets, and the client-side web application visualizes the public mood. The mood visualization consists of the Russian mood geo chart, the mood changes plot through the day, and the mood changes plot through the week.**

Keywords — sentiment analysis; public mood; mood patterns; twitter; social media.

I. Introduction

With the popularization of social media, particularly the micro-blogging website Twitter, a vast amount of content is directly generated by people every day. In addition to textual information, which seems to have affective component, Twitter messages are also time-stamped and geo-located. Consequently, both tweets meaning and extended information about a message can be analyzed in a purpose of scientific studies in general and in the exploration of public mood variations particularly.

In data mining, the usage of social media to analyze and predict political events is becoming more popular in recent times. During the Brexit referendum in the United Kingdom, the researchers consider changes to the public mood within the contents of Twitter [13]. They measure the appearance of positive and negative affect in various geographic regions of the United Kingdom, at hourly intervals. According to the results, there are three key times in the period leading up to and including the EU referendum, each of which was characterized by an increase in negative affect with a corresponding loss of positive affect.

The paper [12] describes an empirical study of Relationship between Twitter mood and stock market from an Indian context. Using Twitter as a source of the news, the authors have extracted the polarity of messages and have found a significant correlation with stock market movement measured in the major stock indices of India. In addition, the correlation of the sentiment with other macroeconomic factors like Gas and Oil Price was established.

Academics from the University of Bristol have published two papers with analysis of periodic patterns in daily media content and consumption under the ThinkBIG project [17]. The first paper [6] was focused on the scrutiny of 87 years of the United States and United Kingdom newspapers between 1836 and 1922. Studies have found people's behavior were strongly correlated with the weather and seasons. In the second paper [7], presented at 2016 IEEE International Conference on Data Mining, the authors pay their attention to discovering mental health changes. The team analyzed Twitter content in the United Kingdom and Wikipedia access over four years using data mining and sentiment analysis techniques. They found that negative sentiment tends to be overexpressed in the winter with the peak value in November, while more aggressive emotions like anxiety and anger seem to be overexpressed between September and April. To conclude, both papers states that people's collective behavior follows strong periodic patterns.

This paper describes the development of the program for monitoring peoples' mood through Twitter content in Russian. This paper aims at implementing the software product for exploring the temporal and geographical mood patterns in Russia using machine learning techniques. In contrast with issues mentioned above, this program is designed to process Twitter data in the online mode, i.e. to receive data directly from Twitter API in real time, rather than analyze the pre-collected messages corpus.

The paper is organized as follows. In section 2 the program implementation, methodology, and data collection are described. Section 3 is focused on results and further ways of research. The limitations of this paper are provided in section 4.

II. Implementation, Data, and Methodology

The client-server model was implemented for this project, where the server-side application collects and analyzes Twitter content, and the client-side web application visualizes results. Python was selected as a preferred programming language because of its cross-platform operability, open source code and a vast number of third-party libraries. The Google App Engine [9] cloud platform was used to run and host this project on Google's infrastructure in Python runtime environment. The applications data is stored in Google App Engine Cloud Datastore [4], that is, a high-performance NoSQL document database.

Fig. 1 illustrates the process of public mood monitoring; it's clear that it can be divided into several parts. Firstly, messages obtained via Twitter API [19] using Python-based client library. Secondly, the identification of a federal subject for each obtained message is performed. Thirdly, sentiment analysis is executed. Fourthly, the information of emotional polarity of messages is stored in the database. At the last step, the client-side application visualizes results. Details for these parts are given in the following sections.

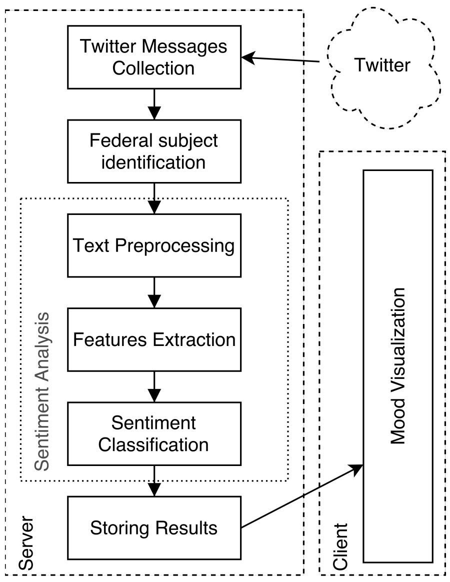


Fig. 1. The program architecture

A. *Twitter messages collection*

Twitting with a location is the geotagging feature in the Twitter platform. On the one hand, this feature helps to provide more meaningful experience for users by making messages more contextual. On the other hand, it makes possible for researchers to analyze Twitter content from the location-based point of view. In order to use the Tweeting with location feature users must opt-in, i.e. turn location "on". The location will be displayed with users Tweets only in case if they give explicit permission for location extraction. Twitter tracks their location via mobile geo-services or IP.

It's common for IT companies to release its API to the public so that other software developers can design products that are powered by its service. To access Twitter content programmatically it's necessary to register the developer application in Twitter Developers Console. Using credentials from the registered application it's possible to interact with Twitter API from the code of the program. The open-sourced library Tweepy [18] was used in this project to communicate with the Twitter platform and use its' API. The cron job, that is, time-based job scheduler in Unix-like computer OS, is searching and collecting new tweets in Russian with geotagging information via Tweepy every minute. In other words, the information about newly published messages is updated in the program every minute.

B. *Federal subject identification by message coordinates*

For each message collected at the previous step the administrative-territorial entity should be defined according to ISO 3166-2:RU standard, that is, part of ISO 3166 standard published by the International Organization for Standardization, which describes the principal subdivisions of all countries coded in ISO 3166-1.

Due to high implementation complexity, it was decided to use existing geographical services to identify federal subjects' codes. The GeoNames [8] worldwide geographical database was selected for identification of the federal subject code by message latitude and longitude values. This service provides developers with HTTP REST API, which includes identification of the country ISO code and the administrative subdivision of any given point. According to GeoNames terms and conditions of use, there are 30000 requests daily limit and 2000 hourly limit for the code identification functional.

C. *Sentiment Analysis*

The sentiment analysis process can be divided into three steps. At the first step, text preprocessing for collected messages is executed to prepare textual information for sentiment analysis. At the second step, classification features are extracted from prepared messages. At the last step, sentiment classification for each message is performed. The detail description of the steps is as follows.

1) *Text preprocessing*

Texts generated by humans in social media sites contain lots of noise that can significantly affect the results of the sentiment classification process. Moreover, depending on the features generation approach, every new word seems to add at least one new dimensional, that makes the representation of texts is sparse and high-dimensional, consequently, the task of the classifier has become more complex. According to [10], text preprocessing has been found crucial on the sentiment classification performance.

To prepare messages, such text preprocessing techniques as reverting repeated letters, removing URLs, removing numbers, converting to lowercase, word normalization and stemming were used in this program. Removing and replacing tasks was performed using regular expressions. The morphological analyzer PyMorphy2 [11]was used for words normalization. Stemming of normalized words was performed using NLTK Python library [16].

2) *Features extraction*

A basic step for a static natural language processing task tends to be the conversion of raw text into features, which provides a machine learning model with a simpler, more comprehensible view of the text. The bag-of-words model was used to calculate texts embedding using unigrams and bigrams.

3) *Sentiment classification*

In this project, the multinomial Naïve Bayes classification algorithm for binary sentiment analysis task was used because of its tendency to perform significantly well in the texts classification task and wide usage [20], [2], [14]. The basic idea of Naïve Bayes technique is to find the probabilities of classes assigned to texts by using the joint probabilities of words and classes [5]. Consider the given data point *x* and class $c \in C$. The starting point is Bayes' theorem for conditional probability which estimates as follows:

$$P(c|x) = \frac{P(X|C)}{P(x)} \quad (1)$$

$$P(x|c) = \frac{count(x,c)}{count(c)} \quad (2)$$

Where *count(x, c)* is the count of word *x* in class *c; count(c)* is a count of all words in class *c*. For texts with unknown words, the estimation (2) might be problematic because it would give zero probability. The usage of Laplace smoothing is a common way to solve this problem (3).

$$P(x|c) = \frac{count(x,c)+1}{count(c)+|V|+1} \quad (3)$$

Where $|V|$ is the length of vocabulary in training set.

From the assumption of word independence, it appears that for data point $x = \{x_1, x_2, ..., x_i\}$ the probability of each of its features to occur in the given class is independent. Thus, the estimation of this probability can be calculated as follows:

$$P(c|x) = P(c) \prod P(x_i|C) \quad (4)$$

In this context, that means the final equation for the class chosen by a naive Bayes classifier is (5).

$$c_{nb} = \underset{c \in C}{\operatorname{argmax}} P(c) \prod P(x_i|c) \quad (5)$$

To avoid underflow and increase speed, the Naive Bayes calculations are performed in the log space (6).

$$c_{nb} = \underset{c \in C}{\operatorname{argmax}} (\log \mathrm{P(c)} + \sum \log P(x_i|c)) \quad (6)$$

The Naive Bayes classifier was trained on the corpus of short texts in Russian based on Twitter messages [3], which consists of 114991 positive and 111923 negative tweets. The 10-fold cross-validation shows accuracy up to 83%.

D. *Storing results*

Every time the cron job have been executed, the new information about publication time, the amount of positive and negative messages for each federal subject is stored in the database.

E. *Visualization*

To explore temporal public mood variations and location based mood values the website was implemented. Both types of graphics were developed with the framework Google Charts [1], which provides developers with the tool for constructing interactive charts for browsers and mobile devices. There are three graphics displayed at the website. The first one is the Russia mood geo chart, where the current mood state for each federal subject is visualized. The second one and the third one are temporal mood changes plots through the day and through the week respectively.

1) *Mood variations*

The information about the time of the day and day of the week is extracted from messages to calculate temporal mood changes. Next, the public mood changes are calculated using the following equitation:

$$mood_t = \frac{pos_t}{pos_t + neg_t} \quad (7)$$

Where, pos_t is the number of positive messages in the specific period *t*; neg_t is the number of negative messages in the specific period *t*. The temporal mood changes chart through the day and through the week are plotted in the program. These charts are constructed over all data that have been processed by the program already, so the level of its accuracy and reliability increases with the number of analyzed tweets.

2) *Mood geo chart*

To plot the mood geo chart, for each federal subject the mood values are calculated using (7) for the last hour. Next, the federal subjects in the geo chart are marked with colors from green to red, where green color means the predominance of positive tweets; yellow color means the balance between the amount of positive and negative messages; red color means the predominance of negative tweets. Fig. 2 illustrates the example of the public mood geo chart for Russia.

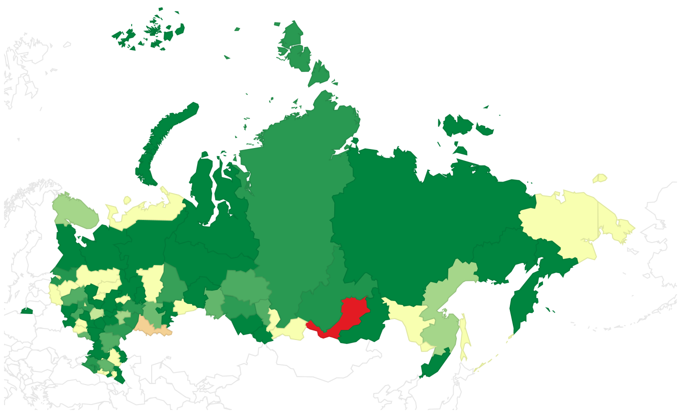


Fig. 2. Example of the public mood geo chart for Russia

III. Results

As a result, the program for public mood monitoring through Twitter content in Russian is implemented as web-service, which can be found by the URL http://twittermood-ru.appspot.com/. The program collects new messages, which are published on Twitter, in real time mode, performs sentiment analysis, process the data obtained at the previous step, and visualizes the results. The mood geo chart provides with an opportunity for monitoring mood values in different regions of Russia for the last hour. The other plots offer valuable insights about temporal public mood changes based on all collected data.

The further research will be focused on extending of analyzed feelings, that means, monitoring not only positive or negative sentiment expressions, but also the expression of fear, sadness, joy, and anger. In addition, the multiclass sentiment classification can be implemented to enhance the quality of public mood calculations.

IV. Limitations

Despite a wide range of Twitter content analysis benefits, it also has some drawbacks. Technically, Twitter users are not representative of the public, consequently, tweets are not representative of the public opinion [15]. Findings in this article apply only to the population of Twitter users geo-located in the Russia. In this work, it's possible to make claims only about the population of Russia Twitter users and not the general population.

References

[1] "Charts | Google Developers," *Google Developers*. [Online]. Available: https://developers.google.com/chart/. [Accessed: 18-Mar-2017].

[2] R. Collins, D. May, N. Weinthal, and R. Wicentowski, "SWAT-CMW: Classification of Twitter Emotional Polarity using a Multiple-Classifier Decision Schema and Enhanced Emotion Tagging," *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 2015.

[3] "Corpus of short texts in Russian," *Julia Rubtsova*. [Online]. Available: http://study.mokoron.com/. [Accessed: 18-Mar-2017].

[4] "Datastore - NoSQL Schemaless Database | Google Cloud Platform," *Google Cloud Platform*. [Online]. Available: https://cloud.google.com/datastore/. [Accessed: 18-Mar-2017].

[5] L. Dey, S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment Analysis of Review Datasets Using Naïve Bayes' and K-NN Classifier," *International Journal of Information Engineering and Electronic Business*, vol. 8, no. 4, pp. 54–62, Aug. 2016.

[6] F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Discovering Periodic Patterns in Historical News," *Plos One*, vol. 11, no. 11, 2016.

[7] F. Dzogang, T. Lansdall-Welfare, and N. Cristianini, "Seasonal Fluctuations in Collective Mood Revealed by Wikipedia Searches and Twitter Posts," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.

[8] "GeoNames," *GeoNames*. [Online]. Available: http://www.geonames.org/. [Accessed: 18-Mar-2017].

[9] "Google App Engine Documentation | App Engine Documentation | Google Cloud Platform," *Google Cloud Platform*. [Online]. Available: https://cloud.google.com/appengine/docs/. [Accessed: 18-Mar-2017].

[10] E. Haddi, "Sentiment analysis: text, pre-processing, reader views and cross domains," dissertation, 2015.

[11] M. Korobov, "Morphological Analyzer and Generator for Russian and Ukrainian Languages," *Communications in Computer and Information Science Analysis of Images, Social Networks and Texts*, pp. 320–332, 2015.

[12] S. Kumar, S. Maskara, N. Chandak, and S. Goswami, "Empirical Study of Relationship between Twitter Mood and Stock Market from an Indian Context," *International Journal of Applied Information Systems*, vol. 8, no. 7, pp. 33–37, 2015.

[13] T. Lansdall-Welfare, F. Dzogang, and N. Cristianini, "Change-Point Analysis of the Public Mood in UK Twitter during the Brexit Referendum," *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, 2016.

[14] B. Le and H. Nguyen, "Twitter Sentiment Analysis Using Machine Learning Techniques," *Advanced Computational Methods for Knowledge Engineering Advances in Intelligent Systems and Computing*, pp. 279–289, 2015.

[15] A. Mitchell and P. Hitlin, "Twitter reaction to events often at odds with overall public opinion," *Pew Research Center*, vol. 4, 2013.

[16] "Natural Language Toolkit," *Natural Language Toolkit — NLTK 3.0 documentation*. [Online]. Available: http://www.nltk.org/. [Accessed: 18-Mar-2017].

[17] "thinkBIG – Patterns in Big Data: Methods, Applications and Implications," *thinkBIG*. [Online]. Available: http://thinkbig.enm.bris.ac.uk/. [Accessed: 18-Mar-2017].

[18] "Tweepy," *Tweepy*. [Online]. Available: http://www.tweepy.org/. [Accessed: 18-Mar-2017].

[19] "Twitter Developer Documentation — Twitter Developers," *Twitter*. [Online]. Available: https://dev.twitter.com/docs. [Accessed: 18-Mar-2017].

[20] Y. Wan and Q. Gao, "An Ensemble Sentiment Classification System of Twitter Data for Airline Services Analysis," *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015.

Narrabat — a Prototype Service for Stylish News Retelling

Irina Dolgaleva, Ilya Gorshkov, Rostislav Yavorskiy
Faculty of Computer Science, Higher School of Economics
Myasnitskaya 20, Moscow, Russia, 101000
E-mail: *iidolgaleva@edu.hse.ru, iagorshkov@edu.hse.ru, ryavorsky@hse.ru*

***Abstract*—Nowadays, news portals are forced to seek new methods of engaging the audience due to the increasing competition in today's mass media. The growth in loyalty of news service consumers may further a rise of popularity and, as a result, additional advertising revenue. Therefore, we propose the tool that is intended for stylish presenting of facts from a news feed. The basic idea is to use a collection of classical literature or poetry as a dictionary of style. The facts are extracted from news texts through Tomita Parser and then presented in the form similar to a sample from the collection. In this framework, we present the current state of Narrabat, a prototype system rephrasing news we are currently working on.**

Keywords: natural language processing, information extraction, natural language generation, tomita parser, neural networks.

I. Introduction

A. The main idea

In the era of information explosion demand for news aggregation services is always high. Classical news services like Yandex News or Google News are on the market for a long time, but their format is too restricted to satisfy all potential audiences. The motivation for Narrabat, a new news service, is to retell news in a stylish way similar to the writings of great writers and poets so as to promote consumers loyalty and to increase revenue of news portals, for instance, from contextual advertising.

The goal of the study is to develop a methodology of rewriting news texts in a specified style and to implement it as a service.

To provide a new insight into retelling news, we build an architecture of Narrabat that is rather straightforward: retrieve news from the providers, extract facts, reproduce the facts in a new form. The realization of the proposed architecture might require to handle two important issues. Firstly, it is necessary to process the news and extract the main information from it. At this point it is essential to realize what kind of unstructured data will be marked as key information. Secondly, we need to generate text in a predefined style considering extracted key words.

To make precise the scope of the study, we explore the methods of retelling the news texts in more capturing manner and build a system that today has no parallel in the integrated marketing communications in news sphere.

The paper presents the current state of the retelling service implementation we are still working on. A well-established result is that we have constructed a prototype system that is capable of producing the poem from the news text. It is to be hoped that in the not too distant future, the findings of current research will be applied to real regularly updated news feed as a service, possibly, as a chat-bot.

The plan of the paper is the following: in section II we present an algorithm of producing poems from the news. In section III the current results are presented. Finally, section IV describes the work still to be done.

B. Related work

Recent years have seen the rapid growth in the number of studies devoted to extraction of information and natural language generation. Insofar as retelling news is concerned to these two subject areas, it would be wise to cover both of them in the paper.

Nowadays, state-of-the-art approaches of fact extraction go far beyond the earliest systems, where the patterns are found referring to rules of grammar [3], [18]. However, an involvement of highly qualified experts in the field or linguists is believed to be a significant drawback of these approaches. Some of them are briefly recalled in the next few paragraphs.

The next coherent idea about highlighting the facts from the text was to propose an algorithm that was able to be trained independently or ”almost independently”, namely, using active learning techniques [13], [9].

As the task of the researches became more complicated, and the need to distinguish an implicitly expressed meaning occurred, the aforementioned approaches lose its efficiency. And the researches shifted their attention to generative models [25] and conditional models [23].

Shedding light on the text generation approaches, the first things that arises is that text in natural language may be generated via predetermined rules [8], [20], when a set of templates is composed to map semantics to utterance. This approach is supposed to be conventional one. These systems are believed to be simple and easy to control, however, at the same time, no scalable due to limited number of rules, and, consequently, output texts.

Furthermore, utilization of statistical approaches in sentence planning are still based on hand-written text generators, whether choosing the most frequent derivation in context-free grammar [5] or maximising the reward in reinforcement learning [28]. By the way, further researches are aimed at minimising human participation and rely on learning sentence planning rules from labelled corpus of utterances [26], which also require a huge markup by linguists.

The next set of approaches in natural language generation is based on corpus-driven dependences. The systems in this direction imply the construction of class-based n-gram language model [24] or phrase-based language model [14]. Moreover, a significant number of researchers utilize active learning in order to generate texts [2], [11].

The use of neural network-based approaches in natural language generation is still relatively unexplored. Although, there are studies that present the high-quality recurrent neural network-based language models [17], [16] that are able to model arbitrarily long dependencies. In addition, it is worth emphasising that the usage of Long Short-term Memory (LSTM) network may try to solve the the vanishing gradient problem [6] such as in [28].

II. Data and Method

A. The news sources

In this framework, we utilize short news texts that were extracted from Russian-language informational portal "Yandex.News". The collection of news consists of 330 texts on different topics, for instance, society, economy, policy, to name but a few (ultimately, 22 topics). This collection of news texts was composed of texts on diverse topics wilfully so as to consider all lexical, syntactic and morphological particularlities of each of the themes in order to create universal system of text processign and generation.

Every text in the collection comprises no more than three sentences except a title. It is worth emphasizing that the format of short texts leads itself well with highlighting the main information from the text. It follows from the fact that every sentence is quite informative to extract key knowledge by means of rule-based approach.

B. Fact extraction

To provide basic information from the news, we propose to extract a kind of extended grammatical basis of the sentences. To that end, we use Tomita-parser [12] that allows to extract structured data (facts) from text in natural language. The tool is much more flexible and effective in key information detection and extraction than, for example, metric *tf-idf* since it allows to retrieve finite chains of words from all the positions in the sentence, not only successive words.

Open-source Tomita-parser, in contrast to similar non-commercial fact extraction software, accounts for specificity of work with the Russian language and has more or less detailed documentation. The tool was implemented by developers of Yandex on the basis of GLP-parser [27], which utilizes context-free grammars, dictionaries of keywords and interpretator.

To get a new insight into extracting the meaning of the texts, a dictionary (gazetteer) and grammar was compiled. As mentioned before, we suggest that the main idea of the sentence is fixed in common basis of the sentence, a kind of analogue of the grammatical basis. Given the opportunity to construct Russian-language sentences with the inversion, the grammar consists of the two main rules:

$$S \rightarrow Subject\ Predicate \mid Predicate\ Subject$$

Every nonterminal derives a string of words dependent on the root words, namely, for *Subject* it may be adjective and for *Predicate* it may be addition or adverb.

After the required string of words is found, Tomita-Parser transforms it into fact and represents it in the result collection of labelled texts, which, in turn, is prepared for text retelling.

C. Poems collection

To teach our system the poetry style we have used writings of Alexander Blok [7] and Nikolay Nekrasov [19] retrieved from Maksim Moshkov on-line library Lib.Ru [1]. We have chosen to utilize particularly these poets as their poems possess artistic and rhythmic harmony, and clearly traceable metrical feet. In further work we plan to expand the collection of poetry by Agniya Barto, Athanasius Fet and Fedor Tyutchev.

D. Learn and produce methods

Besides the method that is described below, we tested another ways of generating word sequences, such as neural networks. For example, we trained a network with LSTM-layer which was expected to generate poems, using a huge dataset of Pushkin's poems from [22]. (LSTM for generating poems was successfully applied in [21], [29], [15]). The result we got was a bit insufficient due to low computational power of our computer and small network size. Further implementations with additional layers increased the quality of generated poems, but it is still being trained, so we are not ready yet to present its results.

Table I presents the example of quatrain generated by the first version of our neural networks:

Narrabat output v.00	Ко в жаме стрьк иреланье, И сталили пореланье И по почаль в сореннем По сеанно переланий.

Table I
The poem produced by neural networks

On the Table I it could be seen that although the poem consists of non-existent Russian-language words, the strings of characters in words virtually resemble real words in their structure. The second thing to sharpen the issue addressing the table is that three out of four strings in the quatrain have the same number of syllables (while the fourth line has only one syllable less). The makings of the rhythms, as well, are evident. Given all the above, we treat the neural networks as a paramount direction for our further research.

E. Current version of the algorithm

Apart from training neural networks to generate poems, we are so far to seek the most conspicuously well-turned poem generator. To that end, we use template-base method described below.

First, in order to break words into syllables, we utilize an improved version of an algorithm of P. Hristov in the modification of Dymchenko and Varsanofiev [4] that comprises a set of syllabication rules that are applied sequentially.

Then syllables of potentially matching subjects and predicates are compared using the following heuristic:

- The number of syllables must coincide.
- Vowels inside syllables have priority over consonants.
- The last syllable has priority over the other.

Search for the similar sentences returns pieces of classical writings, which are used then as templates for the resulting text generation. The output poems ought to be sought in the Section 3.

III. Results

Below is an example produced by current release (v.01) of our Narrabat system. We start from a news description and extract subject and predicate, see Table II.

Source	Общегородской субботник пройдет в следующую субботу, 15 апреля.

Extracted basis	Subject = общегородской субботник
	Predicate = пройдет

Table II
Original news

The same is done for all sentences in the collection, see example in Table III.

Source	А виноградные пустыни, Дома и люди — всё гроба. Лишь медь торжественной латыни Поет на плитах, как труба.

Extracted basis	Subject = медь торжественной латыни
	Predicate = поет

Table III
Original piece from the collections

The implemented similarity measure allows us to figure out that the subjects and the predicates are quite similar, see Table IV. Notice the same number of syllables and almost identical endings.

Subjects	**Predicates**
медь тор-жест-вен-ной ла-ты-ни	по-ет
об-ще-го-род-ской суб-бот-ник	прой-дет

Table IV
Example of a similar pairs match

Now we can replace the matching pairs, see Table V for an example of the resulting poem.

Narrabat output v.01	А виноградные пустыни, Дома и люди — всё гроба. Лишь общегородской субботник Пройдёт на плитах, как труба.

Table V
The final result of the algorithm

One can see that the resulting text keeps subject and predicate from the original fact and at the same time the inserted fragment smoothly fits the style of the poem and do not destroy its structure.

All readers are able to have a closer look at the details of implementation of our Narrabat system and access the source code that is open and available on GitHub [10].

IV. Conclusion and future research directions

In the paper we have proposed a prototype of system that is capable of retelling the news as poems that resembles style of great writers.

In the course of the work we discovered that the collection of poems have to be drastically enlarged in order to generate high-quality poems. Given the exploration, Nikolay Nekrasov has demonstrated more mapping potential in our tasks as he wrote more common sentences than Alexander Blok.

Moreover, the aforementioned metrics of mapping the subjects and predicates from news and poems does not cover all cases to be universal, for instance, in further releases it may take into account rhyme explicitly.

Although the first results presented above are somehow promising, still a lot is on the to do list:

- Improve the quality fact extraction by extending the parsing rules.
- Use available dictionaries of accentuation to take into account the rhythmic structure of a sentence.
- Apply machine learning techniques to better grasp the style of a sample writing.
- Extend the algorithm to cover other parts of sentences, namely, objects and complements.

References

[1] Lib.ru: Library of maksim moshkov. http://lib.ru/. Accessed: 2017-04-10.

[2] Gabor Angeli, Percy Liang, and Dan Klein. A simple domain-independent probabilistic approach to generation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 502–512. Association for Computational Linguistics, 2010.

[3] Douglas E Appelt, Jerry R Hobbs, John Bear, David Israel, and Mabry Tyson. Fastus: A finite-state processor for information extraction from real-world text. In *IJCAI*, volume 93, pages 1172–1178, 1993.

[4] Yura Batora. Algorithm for splitting words into syllables. https://sites.google.com/site/foliantapp/project-updates/hyphenation. Accessed: 2017-04-10.

[5] Anja Belz. Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Natural Language Engineering*, 14(04):431–455, 2008.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[7] Aleksandr Blok. *Sobranie sochinenij v 8 tomah (in Russian)*. Gosudarstvennoe izdatel'stvo hudozhestvennoj literatury, Moscow, 1960-1963.

[8] Adam Cheyer and Didier Guzzoni. Method and apparatus for building an intelligent automated assistant, March 18 2014. US Patent 8,677,377.

[9] Aidan Finn and Nicolas Kushmerick. Active learning selection strategies for information extraction. In *Proceedings of the International Workshop on Adaptive Text Extraction and Mining (ATEM-03)*, pages 18–25, 2003.

[10] Rostislav Yavorskiy Irina Dolgaleva, Ilya Gorshkov. Narrabat. https://github.com/onobot/allbots/tree/master/Ono11_Poems, 2017. Accessed: 2017-05-14.

[11] Ravi Kondadadi, Blake Howald, and Frank Schilder. A statistical nlg framework for aggregated planning and realization. In *ACL (1)*, pages 1406–1415, 2013.

[12] Yandex LLC. Tomita-parser tool to extract structured data from texts. https://tech.yandex.ru/tomita/. Accessed: 2017-04-10.

[13] François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1552–1561. Association for Computational Linguistics, 2010.

[14] François Mairesse and Steve Young. Stochastic language generation in dialogue using factored language models. *Computational Linguistics*, 2014.

[15] Yejin Choi Marjan Ghazvininejad, Xing Shi and Kevin Knight. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, 2016.

[16] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[17] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[18] R Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, volume 328, page 334, 1999.

[19] Nikolaj Nekrasov. *Polnoe sobranie stihotvorenij N. A. Nekrasova v 2 tomah (in Russian)*. Tipografija A. S. Suvorina, Sankt-Peterburg, 1899.

[20] Hugo Gonçalo Oliveira and Amílcar Cardoso. Poetry generation with poetryme. In *Computational Creativity Research: Towards Creative Machines*, pages 243–266. Springer, 2015.

[21] Anna Rumshisky Peter Potash, Alexey Romanov. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924, 2015.

[22] Aleksandr Pushkin. *Sobranie sochinenij v desyati tomah. Tom vtoroj. Stihotvoreniya 1823-1836 (in Russian)*. 1823-1836.

[23] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine learning*, 34(1-3):151–175, 1999.

[24] Adwait Ratnaparkhi. Trainable approaches to surface natural language generation and their application to conversational dialog systems. *Computer Speech & Language*, 16(3):435–455, 2002.

[25] Kristie Seymore, Andrew McCallum, and Roni Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI-99 workshop on machine learning for information extraction*, pages 37–42, 1999.

[26] Amanda Stent and Martin Molina. Evaluating automatic extraction of rules for sentence plan construction. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 290–297. Association for Computational Linguistics, 2009.

[27] Masaru Tomita. Lr parsers for natural languages. In *Proceedings of the 10th International Conference on Computational Linguistics and 22nd annual meeting on Association for Computational Linguistics*, pages 354–357. Association for Computational Linguistics, 1984.

[28] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[29] Rui Yan. i, poet: Automatic poetry composition through recurrent neural networks with iterative polishing schema. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 2238–2244, 2016.

The Program of Syntax-Based Sentiment Analysis

Sergey Smetanin
Faculty of Computer Science
National Research University Higher School of Economics
Moscow, Russia
sismetanin@gmail.com

Abstract — **The popularization of Web 2.0 significantly increased online communications. As a consequence, it provoked the rapid development research in the field of natural language processing in general and sentiment analysis in particular. Information overload and the growing volume of reviews and messages facilitated the need for high-performance automatic processing methods. This paper aims to explore the opinion mining approach, which is based on syntactic dependencies between semantic units as one of the key features of the sentiment classification. Machine learning techniques are widely used in the proposed approach at each step. As a result, a prototype of the program is expected to be implemented. In future work, aspect-based analysis might be considered.**

Keywords—sentiment analysis; machine learning; natural language processing; syntax.

I. INTRODUCTION

Proliferation of the Internet, the increased level of online communication facilitated the rapid development of natural language processing research in general and sentiment analysis in particular. One of the main reasons for this appears to be the popularization of web 2.0 sites, especially social media networks and microblogging services, e.g. Facebook, Twitter, LinkedIn. While using the above-mentioned online social media, users frequently share their emotional states and opinions on a variety of topics. Due to the vast amount of text information, such as reviews, ratings, and messages, human processing has become a challenge, hence a need for automated processing methods emerged.

Automated opinion mining is widely used to determine user attitude to the subject of study, calculate the potential products and services [2], determine the possible future value of stock market objects, and even predict election results. For example, researchers from Northeastern University and Harvard University, studying the characteristics and dynamics of Twitter, have observed a multiple number of trends related to the emotional state in the processed data [3].

Depending on the specifics of the analyzed data, different approaches to the opinion mining may be used. For example, texts from social networks tend to contain spelling and punctuation errors, misspellings, slang, and emoticons. In this case, the algorithm should be resistant to the peculiarities of writing and to using those features for classification, which will help to overcome this problem. Thus, considering the specific characteristics of the data and the usage of efficient features, the high accuracy of analysis can be achieved.

This paper aims to present the proposed approach of opinion mining based on syntactic dependencies between semantic units (e.g. words and phrases) as one of the key features of sentiment analysis. The training data for the classifier were collected from the open-source online resource KinoPoisk [1] with the embedded spelling and punctuation moderation in order to avoid the incorrect semantic patterns extraction. As a result, the program of a high accuracy sentiment analysis based on syntactic dependencies trees will be designed.

The rest of the article is organized as follows. Section 2 makes emphasis on the overview of the related work. Section 3 describes the proposed approach in a context of a phased solution of the opinion mining key tasks in details. By the end, the expected results are mentioned and perspectives of the future research are discussed.

II. RELATED WORK

Opinion mining tends to be an efficient tool not only in the business sector but also in social studies. As a consequence, there are many services that have the various power and functionality. The detailed description of the most similar projects is following.

The first one is Bluemix by IBM company. This is a cloud platform aimed to develop, launch and administrate cloud web applications. One of the key features of this platform is Twitter messages sentiment analysis, which provides 4 class (i.e. positive, negative, neutral, and ambivalent) classification in English, Spanish, German, and French languages. According to the terms of use [5], it is free for platform users to analyze up to 5 million of tweets per month, and if the limit is exceeded, they ought to separately pay for each additional million of processed messages.

It is clear that the language support is critical for the natural language processing, therefore, Bluemix is not suitable for the Russian texts because of the lack of the analyzed language support. In addition, the restriction on the source of processing messages (only from Twitter) makes the service extremely limited in functionality and capabilities. In contrast, the program proposed in this paper is concentrated at Russian language linguistic features in order to achieve high-quality results; consequently, it has a competitive advantage.

The next one is Fact Factor SDK by RCO company. This is a program library that provides a high-quality sentiment analysis on Russian including part of speech tagging, syntactic relation extraction and linguistic style identification [6]. The versatile analysis of the text makes it a powerful tool not only in the field of opinion mining but also in other applications of natural language processing. The SDK is distributed as paid software in three possible assemblings with different functionality. However, a trial version can be obtained after a license agreement conclusion.

The main disadvantage of this solution seems to be the absence of a graphical interface that makes the product is available for use only for software developers. In this case, the proposed approach has an essential advantage, namely user-friendly web-interface.

The last one is Eureka Engine, a highly-optimized service of linguistic analysis, which allows to identify the tonality of the opinion, extract facts and pieces of knowledge from raw data. It consists of 6 modules that solve tasks from subjects extraction to parts of speech tagging. It was developed by PalitrumLab, one of the most sought companies at the data mining market in Russia. According to the price list [7], it offers services for a monthly paid subscription, but a free trial version for 2 months can be requested. However, even with the good quality of provided functionality, this service seems to be available for companies primarily because of high costs on usage.

III. Methodology

The program structure is designed as a combination of the 3 extensive subtasks. To start with, segmentation of a text into sentences is performed. Next, a tree of syntactic dependencies is extracted from sentences obtained in the previous step. At the last stage, the program provides a binary (i.e. 'positive' and 'negative' labels) classification of the tonality based on the extracted features vector with additional syntactic features. The detailed description of the approach to each is the following.

A. Sentence boundary disambiguation

According to punctuations rules in the Russian language, the terminal marks (i.e. dot, question mark and exclamation mark) do not always indicate the end of the sentence. For example, the point is usually used in URL-links, after the initials, in the notation of date and time, and in reductions. Moreover, a considerable emphasis should be taken on the indirect speech punctuation rules, where an indication of the sentence ending in some cases may present a challenge even for a native speaker. Further, as was mentioned above, punctuation errors in texts, which are typical for social networks, complicate the task of sentence boundary disambiguation.

It is important to consider the structural features of NLP. Therefore, scientific researches were purposefully developed for Russian languages and reviewed at the first order. The approach described in [8] was selected as the most suitable for the implementation complexity and accuracy. In general terms, the method proposed in that article consists of free steps. Firstly, each termination mark is signed as a potential ending of the sentence. Secondly, the features vector is extracted from the context of each potential sentence ending. Lastly, the classifier resolve disambiguates in borders of the sentence. As a base classifier algorithms Decision Tree or SVM-light could be used. The last one is able to provide accuracy up to 99.6% mark [8], which is acceptable for the further processing.

B. Synctactic dependencies extraction

It should be noted that the word order in the Russian language significantly influences and defines the meaning and the emotional level of the sentence or phrase. Thus, the use of syntactic dependencies as features allows constructing a more accurate and sensitive to the language system. The syntax structure is based on the concept of phrase, that is, an association of independent words on the basis of subordination syntactic dependency. The words in the phrase are generally linked to the meaning and grammatically. Phrases are not independent syntactic units in speech because they do not convey a complete thought. Therefore, phrases are combined into a sentence via various types of linguistic references thereby forming a complete thought.

In order to remedy the problem of a syntactic dependencies extraction, machine learning methods are frequently used. Firstly, each word in the sentence is determined by the part of speech. If one word corresponds to multiple parts of speech labels, the disambiguation is removed using the algorithm described in [9]. Secondly, feature vectors are extracted from the context of the analyzed text for the further classification. Finally, classifier determines the most probable syntactic connections, thereby identifying the linguistic structure of the sentence. According to [10], accuracy up to 97.18% can be achieved by using such methodology.

C. Sentiment classification

In general, there are three main approaches to the opinion mining problem. The first one called rule-based processes the text according to a set of rules and identifies the opinion. However, this approach is not acceptable, within the scope of this study due to lack of linguistic rules and as a consequence, inability to achieve high-quality results. The next algorithm uses tonal dictionaries. In this case, every word in the text is assigned the value from the dictionary, and then the aggregate function is executed to compute the emotion label. This approach handles the text as an incoherent collection of words and is not sensitive to syntactic structures, which makes it unsuitable for the future use. The last approach is a statistical method based on machine learning technique. This method will be discussed hereinafter.

The result of machine learning methods usage practically depends on the document submission for the classifier, that is, a set of attributes, which are used for the construction of the feature vector. As a rule, the model using bag-of-words and N-grams are used to work with the natural language processing. However, such features as parts of speech, punctuation, syntactic dependencies, and emoticons could significantly improve the outcome score in some cases. In this paper, the architecture of the classifier will be based on the model described in [11], but with additional observing syntactic relations in sentences. Thus, the program obtains the input text, splits it into the sentences, defines the syntax for each sentence and determines the sentiment type.

As a form of interaction with the user, the web-interface will be designed. In the first version, it will contain the text box, as well as a button. When the user will click on the last one, the analysis will take place and the result of processing will be displayed under the text form.

IV. Conclusion

As a result of this research, the prototype of the program of texts sentiment analysis based on syntactic dependencies trees is expected to be implemented. The described structure of the program provides both high-quality classification by using syntactic features and fast performance by implementing the engineered architecture. The output result is represented to the user in a simple and comprehensible form in order to be available for understanding to non-expert users.

The further research will be focused on the user interface improvement and expanding the functionality of the program, some of the key potential features are mentioned below. To start with, aspect-based sentiment analysis, that is, the combination of the both the determination of the opinion relative to the monitored object and identifying its characteristics, can significantly enhance the possible practical applications. For example, this feature affords not only to determine the attitude of customers to the product through the processing of their reviews from social networks, but also to identify the strong and weak properties of monitored product. Next, the subjectivity classification, which the main goal is concentrated on establishing whenever the text is subjective or objective makes, could be added. In this case, a 3-way classification can be achieved because objectivity reviews contain only a description of the facts and can be marked as neutral. As for the design, a user-friendly interface with the ability of the multiple number texts processing at the one time will be implemented. Thus, the complex upgrade of the program will be achieved.

References

[1] Kinopoisk [Online]. Available: https://www.kinopoisk.ru/. [Accessed: 30- Jan- 2016].

[2] K. Bauman, B. Liu and A. Tuzhilin, "Recommending Items with Conditions Enhancing User Experiences Based on Sentiment Analysis of Reviews", Proceeding of CBRecSys, 2016, p. 19.

[3] T. Nguyen, K. Shirai and J. Velcin, "Sentiment Analysis on Social Media for Stock Movement Prediction", Expert Systems with Applications, vol. 42, no. 24, pp. 9603-9611, 2015.

[4] Northeastern University, "Pulse of the Nation: U.S. Mood Throughout the Day inferred from Twitter", 2011. [Online]. Available: http://www.ccs.neu.edu/home/amislove/twittermood/. [Accessed: 29- Jan- 2016].

[5] IBM - United States, "Insights for Twitter - IBM Bluemix". [Online]. Available: https://console.ng.bluemix.net/catalog/services/insights-for-twitter/. [Accessed: 29- Jan- 2016].

[6] RCO, "RCO Fact Extractor SDK | RCO". [Online]. Available: http://www.rco.ru/?page_id=3554. [Accessed: 30- Jan- 2016].

[7] Eurekaengine, "Eureka Engine". [Online]. Available: http://eurekaengine.ru/price. [Accessed: 30- Jan- 2016].

[8] O. Yrupina, "Detecting Sentence Boundaries in Russian", Computational Linguistics and Intellectual Technologies, vol. 7, no. 14, pp. 539-544, 2008.

[9] A. Sokirko and S. Toldova, "Comparing a Stochastic Tagger Based on Hidden Markov Model with a Rule-Based Tagger for Russian", Ural Federal University, 2005. [Online]. Available: http://elar.urfu.ru/bitstream/10995/1391/1/IMAT_2005_05.pdf. [Accessed: 30- Jan- 2016].

[10] A. Antonova and A. Misyurev, "Russian Dependency Parser SyntAutom at the Dialoge 2012 Parser Evaluation Task", Computational Linguistics and Intellectual Technologies, vol. 11, no. 2, pp. 104-118, 2012.

[11] Y. Adaskina, P. Panicheva and A. Popov, "Syntax-based Sentiment Analysis of Tweets in Russian", Computational Linguistics and Intellectual Technologies, vol. 14, no. 2, pp. 1-11, 2015.

Fast L^1 Gauss 2D Image Transforms

Dina Bashkirova[1,2*], Shin Yoshizawa[1§], Rustam Latypov[2†], Hideo Yokota[1¶]

[1]Image Processing Research Team
RIKEN Center for Advanced Photonics, RIKEN
2-1, Hirosawa, Wako, Saitama, 351-0198, Japan
Email: {*dina.bashkirova,§shin,¶hyokota}@riken.jp

[2]Institute of Computational Mathematics and Information Technologies
Kazan Federal University
420008, Kazan, 35 Kremlyovskaya
Email: †roustam.latypov@kpfu.ru

***Abstract*—Gaussian convolution has many science and engineering applications, and is widely applied to computer vision and image processing tasks. Due to its computational expense and rapid spreading of high quality data (bit depth/dynamic range), accurate approximation has become important in practice compared with conventional fast methods. In this paper we propose a novel approximation method for fast Gaussian convolution of 2D images. Our method employs L^1 distance metric to achieve fast computations while preserving high accuracy. Our numerical experiments show the advantages over conventional methods in terms of speed and precision.**

***Keywords*—Gaussian Smoothing, Laplace Distribution, Fast Approximation Algorithms.**

I. Introduction

Gaussian convolution is a core tool in mathematics and many related research areas, such as probability theory, physics, and signal processing. Gauss transform is a discrete analogue to the Gaussian convolution, and has been widely used for many applications including kernel density estimation [1] and image filtering [2]. Despite its reliable performance and solid theoretical foundations, Gauss transform in its exact form along with other kernel-based methods has a drawback – it is very computationally expensive (has quadratic computational complexity w.r.t. the number of points) and hard to scale to higher dimensions. Which is why there have been many attempts to overcome these problems by creating approximation algorithms, such as fast Gauss transform [3], dual-tree fast Gauss transforms [4], fast KDE [5], and Gaussian kd-trees [6]. Also, box kernel averaging [7] and recursive filtering [8] have been popular in computer graphics and image processing because of their simplicity, see the surveys [9], [10] for numerical comparisons of these approximation methods.

Since high bit depth (also dynamic range) images have become popular in both digital entertainment and scientific/engineering applications, it is very important to acquire high approximation precision and to reduce artifacts cased by drastic truncation employed in many conventional methods focused on computational speed. One of the highly accurate methods is called fast L^1 Gauss transform approximation [11] based on using L^1 distance instead of conventional L^2 Euclidean metric. This L^1 metric preserves most of the properties of the L^2 Gaussian, and is separable, hence it allows to perform computations along each dimension separately, which is very beneficial in terms of computational complexity. Also, L^1 Gaussian has only one peak in Fourier domain at the coordinate origin, and therefore its convolution does not have some undesirable artifacts that box kernels and truncation methods usually have. However, this algorithm works only on one-dimensional (1D) point sets, although it can be extended to uniformly distributed points in higher dimensions by performing it separately in each dimension. In order to be able to acquire Gauss transform for non-uniformly distributed two-dimensional points and to further generalize it to higher dimensional cases, we need to extend existing method [11] to the 2D uniform case.

In this paper we propose a novel approximation method for fast Gauss two-dimensional (2D) image transform. Our method is based on extending the fast L^1 Gauss transform approximation on uniformly distributed 2D points that allows to perform Gaussian convolution quickly while preserving high accuracy. We demonstrate that efficiency of the proposed method in terms of computational complexity, numerical timing, and approximation precision.

II. Fast L^1 Gauss Transform

In this section, we briefly describe the 1D domain splitting algorithm [11] employed for fast L^1 Gauss transforms.

Consider the ordered point set $\mathbb{X} = \{x_i\}_{i=1}^{N}$, $x_i \in \mathbb{R}$, $x_i \geq x_{i-1}$, $\forall i = \overline{2, N}$. Each point x_i has a corresponding value $I_i \in \mathbb{R}$, e.g. pixel intensity in case of images. The L^1 Gauss transform for each point in set $\mathbb{X}$ is given by

$$J(x_j) = \sum_{i=1}^{N} G(x_j - x_i) I_i, \quad G(x) = \exp(-\frac{|x|}{\sigma}), \tag{1}$$

where $G(x)$, $x \in \mathbb{R}$, is a L^1 Gaussian function (also called Laplace distribution in statistics) with its standard deviation σ. It is convenient to decompose L^1 norm by splitting its domain by using the point x_1 such that

$$|x_j - x_i| = \begin{cases} |x_j - x_1| - |x_i - x_1| & \text{if } x_1 \leq x_i \leq x_j, \\ |x_i - x_1| - |x_j - x_1| & \text{if } x_1 \leq x_j \leq x_i. \end{cases} \tag{2}$$

Thus, Gauss transform (1) using the equation (2) becomes

$$\begin{aligned} J(x_j) = I_i + G(x_j - x_1) \sum_{i=1}^{j-1} \frac{I_i}{G(x_i - x_1)} + \\ + \frac{1}{G(x_j - x_1)} \sum_{i=j+1}^{N} G(x_i - x_1) I_i. \end{aligned} \tag{3}$$

Such representation (3) allows to reduce the amount of computational operations, since values $G(x_j - x_1)$, $\frac{1}{G(x_j - x_1)}$, and the sums $\sum_{i=1}^{j-1} \frac{I_i}{G(x_i - x_1)}$ and $\sum_{j+1}^{N} I_i G(x_i - x_1)$ can be precomputed in linear time. However, using the equation (3) may imply some numerical issues, such as overflow, if the distance between x_1 and x_l, $l \in \{i, j\}$ is relatively large. To avoid such issues, this algorithm introduced certain representative points (poles) $\{\alpha_k \in \mathbb{R}\}$ instead of using the single point x_1, where the distance between α_k and x_l is smaller than the length that causes the numerical instability. Hence the equation (3) becomes more complex form, a highly accurate truncation can be applied where $G(\alpha_k - x_j)$ is equal to numerically zero, see [11] for further technical details.

Although this algorithm can be used in case of multidimensional images by applying it separately in each dimension, this separable implementation approach is not applicable to non-uniformly distributed high-dimensional point sets. Therefore we present a novel and natural extension of the domain splitting concept on 2D cases (images) in the following sections.

III. Two-dimensional Algorithm

For a given 2D point set $\mathbb{X} = \{\mathbf{x}_i\}_{i=1}^N$, $\mathbf{x}_i = (x_i, y_i) \in \mathbb{R}^2$, L^1 distance between two points in $\mathbb{R}^2$ is given by $|\mathbf{x}_j - \mathbf{x}_i| = |x_j - x_i| + |y_j - y_i|$, thus the Gauss transform (1) is represented by the formula:

$$J(\mathbf{x}_j) = \sum_{i=1}^{N} \exp\left(-\frac{|x_j - x_i| + |y_j - y_i|}{\sigma}\right) I_i.$$

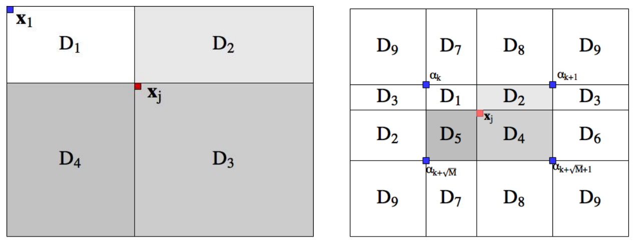
Domain splitting (2) for 2D points is given by

$$|x_j - x_i| + |y_j - y_i| =$$

$$\begin{cases} |x_j - x_1| - |x_i - x_1| + |y_j - y_1| - |y_i - y_1| & \text{if } \mathbf{x}_i \in D_1 \\ |x_i - x_1| - |x_j - x_1| + |y_j - y_1| - |y_i - y_1| & \text{if } \mathbf{x}_i \in D_2 \\ |x_j - x_1| - |x_i - x_1| + |y_i - y_1| - |y_j - y_1| & \text{if } \mathbf{x}_i \in D_3 \\ |x_i - x_1| - |x_j - x_1| + |y_i - y_1| - |y_j - y_1| & \text{if } \mathbf{x}_i \in D_4, \end{cases}$$

$$\begin{aligned} D_1 &= \{\mathbf{x}_i | x_1 \le x_i \le x_j, y_1 \le y_i \le y_j\}, \\ D_2 &= \{\mathbf{x}_i | x_1 \le x_j \le x_i, y_1 \le y_i \le y_j\}, \\ D_3 &= \{\mathbf{x}_i | x_1 \le x_i \le x_j, y_1 \le y_j \le y_i\}, \\ D_4 &= \{\mathbf{x}_i | x_1 \le x_j \le x_i, y_1 \le y_j \le y_i\}, \end{aligned}$$

see Fig. 1a for geometric illustration of the domains.



(a) Single pole $\mathbf{x}_1$ case (b) Multipole $\{\alpha_k\}$ case

Fig. 1: Illustration of 2D domain splliting.

Using the above decomposition, Gauss transform is represented similar to (3):

$$\begin{aligned} J(\mathbf{x}_j) = I(\mathbf{x}_j) + F(x_j)F(y_j) \sum_{\mathbf{x}_i \in D_1(j)} \frac{1}{F(x_i)F(y_i)} I(\mathbf{x}_i) + \\ + \frac{F(x_j)}{F(y_j)} \sum_{\mathbf{x}_i \in D_2(j)} \frac{F(y_i)}{F(x_i)} I(\mathbf{x}_i) + \frac{F(y_j)}{F(x_j)} \sum_{\mathbf{x}_i \in D_3(j)} \frac{F(x_i)}{F(y_i)} I(\mathbf{x}_i) + \\ + \frac{1}{F(x_j)F(y_j)} \sum_{\mathbf{x}_i \in D_4(j)} F(x_i)F(y_i)I(\mathbf{x}_i), \end{aligned} \tag{4}$$

where $F(x_j) \equiv G(x_j - x_1)$ and $F(y_j) \equiv G(y_j - y_1)$.

Precomputation and storage of values $F(x_j)F(y_j)$, $\frac{F(x_j)}{F(y_j)}$, $\frac{1}{F(x_j)F(y_j)}$, and $\frac{F(y_j)}{F(x_j)}$ require $O(4N)$ operations and $O(4N)$ space, and all the subsequent sums can be iteratively computed in $O(N)$ operations. Gauss transform for all points using the formula (4) requires $O(10N)$ as opposed to employing the separable implementation of equation (3) for $O(6N)$ operations. Since computing the Gauss transform using the equation (4) is numerically troublesome, it is reasonable to divide the space into smaller groups and perform computations separately, as it was proposed in [11]. Let us introduce a novel 2D multipole approach for solving this problem.

Consider a set of poles $\{\alpha_k\}_{k=1}^M$, $\alpha_k = (a_k, b_k) \in \mathbb{R}^2$. The distance between points in $\mathbb{X}$ using poles α_k is given by $|\mathbf{x}_i - \mathbf{x}_j| =$

$$\begin{cases} |x_i - a_k| - |x_j - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_1 \\ |x_j - a_k| - |x_i - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_2 \\ |x_i - a_k| + |x_j - a_k| + |y_i - b_k| - |y_j - b_k| & \text{if } \mathbf{x}_i \in D_3 \\ |x_i - a_k| - |x_j - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_4 \\ |x_j - a_k| - |x_i - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_5 \\ |x_i - a_k| + |x_j - a_k| + |y_j - b_k| - |y_i - b_k| & \text{if } \mathbf{x}_i \in D_6 \\ |x_i - a_k| - |x_j - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_7 \\ |x_j - a_k| - |x_i - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_8 \\ |x_i - a_k| + |x_j - a_k| + |y_i - b_k| + |y_j - b_k| & \text{if } \mathbf{x}_i \in D_9, \end{cases}$$

where

$$D_1 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_1^y\}, D_2 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_1^y\},$$

$$D_3 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_1^y\}, D_4 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_2^y\},$$

$$D_5 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_2^y\}, D_6 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_2^y\},$$

$$D_7 = \{\mathbf{x}_i | x_i \in D_1^x, y_i \in D_3^y\}, D_8 = \{\mathbf{x}_i | x_i \in D_2^x, y_i \in D_3^y\},$$

$$D_9 = \{\mathbf{x}_i | x_i \in D_3^x, y_i \in D_3^y\},$$

$$\begin{aligned} D_1^x &= \{x_i | a_k \le x_i \le x_j \text{ or } x_j \le x_i \le a_k\}, \\ D_2^x &= \{x_i | a_k \le x_j \le x_i \text{ or } x_i \le x_j \le a_k\}, \\ D_3^x &= \{x_i | x_i \le a_k \le x_j \text{ or } x_j \le a_k \le x_i\}, \\ D_1^y &= \{y_i | b_k \le y_i \le y_j \text{ or } y_j \le y_i \le b_k\}, \\ D_2^y &= \{y_i | b_k \le y_j \le y_i \text{ or } y_i \le y_j \le b_k\}, \\ D_3^y &= \{y_i | y_i \le b_k \le y_j \text{ or } y_j \le b_k \le y_i\}, \end{aligned}$$

$$J(\mathbf{x}_j) = I_j + \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i \in D_1} \frac{I_i}{\mathcal{G}(x_i)\mathcal{G}(y_i)} + \frac{1}{\mathcal{G}(x_j)\mathcal{G}(y_j)} \sum_{\mathbf{x}_i \in D_5} \mathcal{G}(x_i)\mathcal{G}(y_i)I_i + \frac{\mathcal{G}(y_j)}{\mathcal{G}(x_j)} \sum_{\mathbf{x}_i \in D_2} \frac{\mathcal{G}(x_i)}{\mathcal{G}(y_i)} I_i + \frac{\mathcal{G}(x_j)}{\mathcal{G}(y_j)} \sum_{\mathbf{x}_i \in D_4} \frac{\mathcal{G}(y_i)}{\mathcal{G}(x_i)} I_i +$$
$$+ \sum_{\alpha_k \in D_9} A_k^j + \sum_{\alpha_k \in D_7} B_k^j + \sum_{\alpha_k \in D_8} C_k^j + \sum_{\alpha_k \in D_3} D_k^j + \sum_{\alpha_k \in D_6} E_k^j, \tag{5}$$

$$A_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i=\lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i)I_i, \quad B_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i=\lambda(k)}^{\lambda(k+1)-1} \frac{\mathcal{G}(y_i)}{\mathcal{G}(x_i)} I_i, \quad C_k^j = \frac{\mathcal{G}(y_j)}{\mathcal{G}(x_j)} \sum_{\mathbf{x}_i=\lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i)I_i,$$

$$D_k^j = \mathcal{G}(x_j)\mathcal{G}(y_j) \sum_{\mathbf{x}_i=\lambda(k)}^{\lambda(k+1)-1} \frac{\mathcal{G}(x_i)}{\mathcal{G}(y_i)} I_i, \quad E_k^j = \frac{\mathcal{G}(x_j)}{\mathcal{G}(y_j)} \sum_{\mathbf{x}_i=\lambda(k)}^{\lambda(k+1)-1} \mathcal{G}(x_i)\mathcal{G}(y_i)I_i.$$

see Fig. 1b for geometric illustration of the domains with their poles. The point $\mathbf{x}_j$ is assigned for one representative pole defined by

$$\alpha_k(\mathbf{x}_j) = \max_k \{\alpha_k | a_k \le x_j, b_k \le y_j\},$$

which is the closest pole to $\mathbf{x}_j$ that has absolute values of coordinate smaller than $\mathbf{x}_j$.

For each point $\mathbf{x}_j$, the multipole L^1 Gauss transform is given by the equation (5) where $\mathcal{G}(x_j) \equiv G(x_j - a_k)$, $\mathcal{G}(y_j) \equiv G(y_j - b_k)$, and $\lambda(\cdot)$ is an index function defined by

$$\lambda(k) = \min_{1 \le j \le N} (\mathbf{x}_j | a_k \le x_j < a_{k+1} \text{ and } b_k \le y_j < b_{k+1}).$$

For the sake of simplicity, we assume that the numbers of poles in 2D are same M. Following [11], M and the poles $\{\alpha_k\}$ are given by

$$\{a_k\} = \{b_k\} = \frac{\{0, 1, 2, ..., (M-1)\}w}{M}, \tag{6}$$
$$w = \max(|x_1 - x_N|, |y_1 - y_N|), \ M = [\frac{w}{\varphi\sigma \log(\text{MAX})}],$$

where $[\cdot]$ is the ceiling function, MAX is the maximum value of precision (e.g., double floating point: DBL_MAX in C programming language), and φ is a user-specified parameter (0.5 is employed in our numerical experiments). The above pole selection scheme leads to $\max(G(a_{k+1} - a_k), G(b_{k+1} - b_k)) <$ MAX which theoritically guarantees numerical stability in our method.

If the distance between poles is determined by the equation (6) and $G(\alpha_k - \mathbf{x}_j)$ becomes numerically zero if $|\alpha_k - \mathbf{x}_j| > \frac{w}{\varphi M}$, we can efficiently truncate Gauss transform by approximating the values:

$$\sum_{\alpha_k \in D_9} A_k^j \approx \sum_{\alpha_k \in \mu(D_9)} A_k^j, \sum_{\alpha_k \in D_7} B_k^j \approx \sum_{\alpha_k \in \mu(D_7)} B_k^j,$$
$$\sum_{\alpha_k \in D_8} C_k^j \approx \sum_{\alpha_k \in \mu(D_8)} C_k^j, \sum_{\alpha_k \in D_3} D_k^j \approx \sum_{\alpha_k \in \mu(D_3)} D_k^j,$$
$$\sum_{\alpha_k \in D_6} E_k^j \approx \sum_{\alpha_k \in \mu(D_6)} E_k^j,$$

where $\mu(D_*) = \{\mathbf{x}_i \in D_* \mid |\alpha_k(\mathbf{x}_j) - \alpha_k(\mathbf{x}_i)| \le \frac{w}{\varphi M}\}$. In other words, instead of computing terms $A_k^j, B_k^j, C_k^j, D_k^j, E_k^j$ across all the corresponding point sets, we take into account only the neighbouring points, which allows to avoid nested loop structure in our implementation and speed up the computational process.

As in the 1D algorithm [11], the terms can be iteratively computed in linear time. Assume that an image consists of $\sqrt{N} \times \sqrt{N}$ pixels and the number of poles along each dimension is equal to M, total complexity of our method is equal to $O(16N + 2\frac{\sqrt{N}}{M} + 4\frac{N}{M^2})$, which is a little bit slower than the separable implementation employed in [11] that requires $O(12N + 2\sqrt{N} + M)$ operations.

IV. Numerical Experiments

We held all the experiments on Intel Core i7-6600U 2.60 GHz dual core computer with 16GB RAM and a 64-bit operating system. We compared the multipole version of our algorithm with box kernel (Box) using moving average method [7], the 1D domain splitting (YY14) with separable implementations [11], and Fast Discrete Cosine Transform (FDCT) via the FFT package [12] well-known for its efficiency.

(a) Input image 1

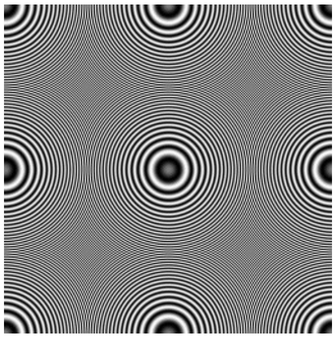
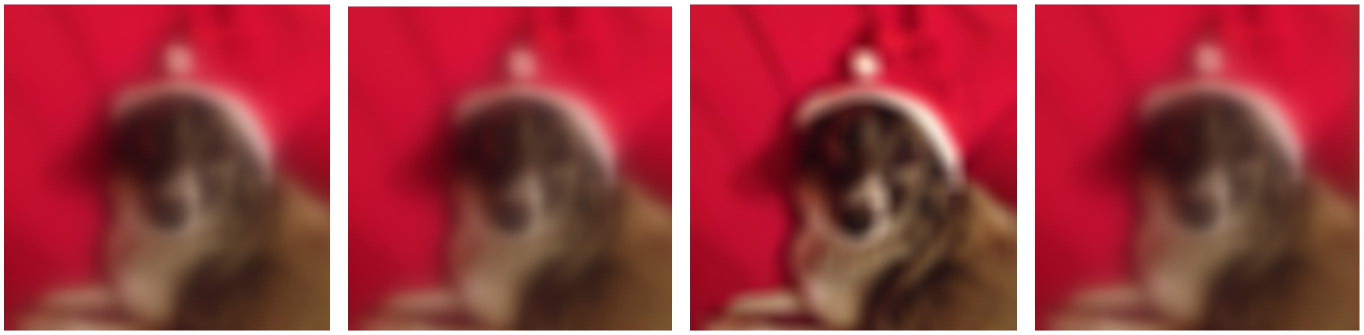

(b) Input image 2

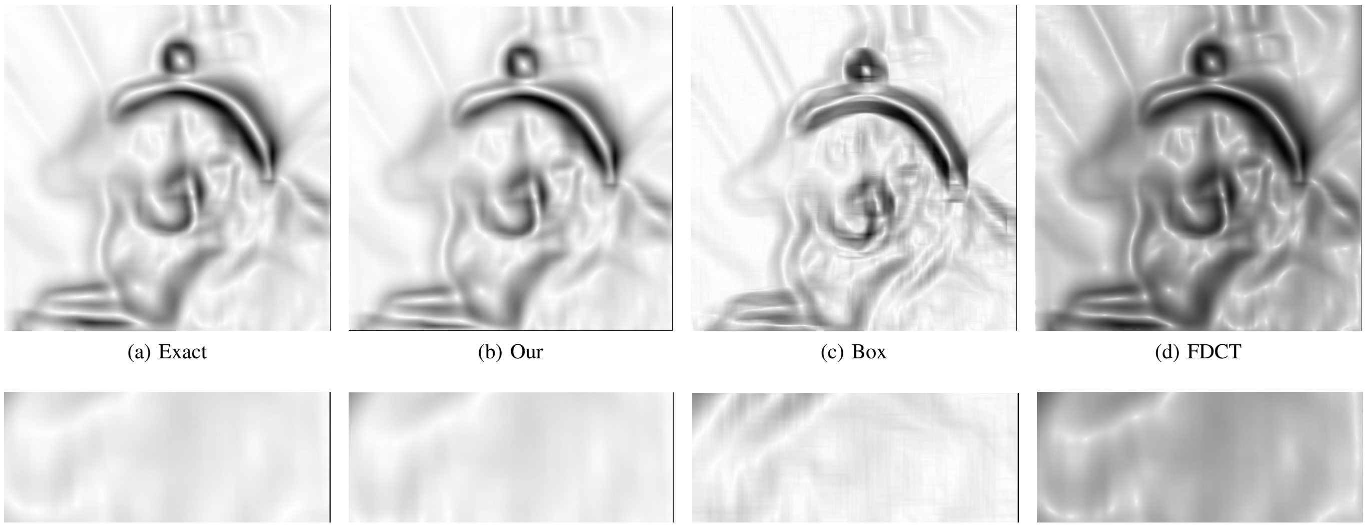
Fig. 2: Input images.

To evaluate the performance of the methods mentioned above we used randomly generated 2D point sets with 10 different sizes from 128^2 to 5120^2 and 10 various values of $\sigma = 5, 10, ..., 50$. The radius for the Box method was chosen equal to σ. The timing results (see Fig. 5) show that our method is slightly slower than the 1D domain splitting (YY14) despite its theoretical complexity is much larger. It is worth noticing



(a) Exact (b) Our (c) Box (d) FDCT

Fig. 3: Results of smoothing ($\sigma = 20$).



(a) Exact (b) Our (c) Box (d) FDCT

(e) Exact (f) Our (g) Box (h) FDCT

Fig. 4: Visualisation of $|\nabla I|$ for comparison of artifacts ($\sigma = 20$).

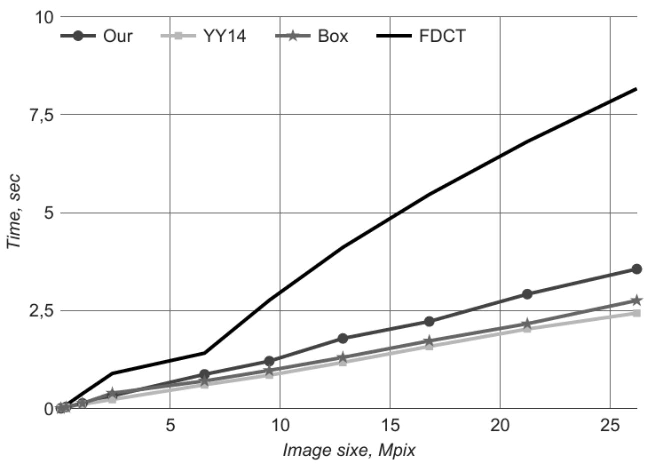


Fig. 5: Timing with respect to image size (averaged by σ).

that the implementation of our method can be further improved by using GPU-based or parallel computing techniques.

However, accuracy evaluation results (see Table I) show that our method achieves best approximation quality among the discussed methods. We evaluate the precision using $E_{\max}$ and PSNR measures. Consider I^e is the exact L^1 Gauss transform result, I^a is the approximation achieved by a given algorithm, and $d_i = |I_i^e - I_i^a|$, $E_{\max}$ is calculated using formula $E_{\max} = \max\limits_{1 \le i \le N} d_i$. We also use peak signal-to-noise ratio (PSNR) [2] to measure the performance of our algorithm according to the equation

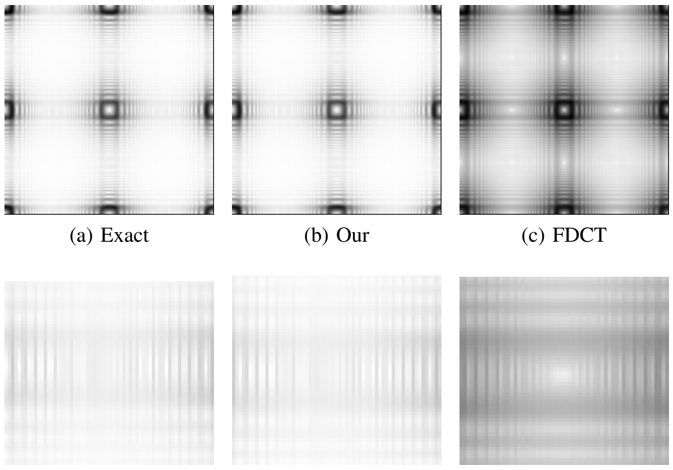
$$\text{PSNR} = -10\log\left(\sum_{i=1}^{N}\left(\frac{d_i}{\max(I_i^e, I_i^a)}\right)^2\right).$$

We performed linear image smoothing by the following normalized convolutions for each color channel:

$$\frac{\int G(\mathbf{x}-\mathbf{y})I(\mathbf{y})d\mathbf{y}}{\int G(\mathbf{x}-\mathbf{y})d\mathbf{y}} \rightarrow \frac{J(\mathbf{x}_j)}{\sum_i^N G(\mathbf{x}_j - \mathbf{x}_i)}$$

TABLE I: Precision and speed evaluation results (speed measured in Mpix/sec).

	Our	**YY14**	**FDCT**	**Box**
$E_{\max}$	$\mathbf{1.8 \times 10^{-11}}$	3.8×10^{-10}	0.44	3.73
PSNR	**291.05**	281.81	58.98	41.45
Speed	7.19	**9.76**	3.37	8.58


(a) Exact (b) Our (c) FDCT

(d) Exact (e) Our (f) FDCT

Fig. 6: Visualisation of $|\nabla I|$ for comparison of artifacts of FDCT ($\sigma = 20$).

where the denominator is also obtained by our method convolving L^1 Gaussian with the image whose intensity is equal to one everywhere.

Fig. 3 illustrates the smoothing results using naive implementation (Exact), our method, Box kernel, and FDCT algorithms. The gradient magnitude $|\nabla I|$ of smoothed images on Fig. 4 and 6 show that, in contrast to FDCT and Box kernel, our method does not produce some undesirable artifacts and is extremely close to the exact implementation.

V. Conclusion

In this paper we presented a novel and fast approximation method for L^1 Gauss 2D image transforms. Series of numerical experiments have shown that our method is generally more accurate than the conventional methods and faster than the widely used FFT. We also demonstrated capability of the proposed method in image smoothing application where the conventional box kernel averaging and FFT both suffer from undesirable artifacts. Despite our method is slightly slower than the separable implementations of 1D algorithm [11], this approach can be efficiently used for non-uniformly distributed points.

Our method is applicable only to uniformly distributed structures, such as images. Hence our future work includes extending the proposed method to higher-dimensional non-uniform cases which can be done for example by using tree-like structures. We also would like to investigate possible applications of the proposed method to various machine learning and image processing tasks, such as regression, segmentation, and registration.

Acknowledgements

This work was supported in part by Grants-in-Aid for Scientific Research of Japan (15H05954 and 16K15019).

References

[1] A. Elgammal, R. Duraiswami, and L. Davis, "Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 25, no. 11, pp. 1499–1504, 2003.

[2] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," in *Proc. of European Conference on Computer Vision (ECCV)*. Springer, 2006, pp. 568–580.

[3] L. Greengard and J. Strain, "The fast Gauss transform," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[4] D. Lee, A. Gray, and A. Moore, "Dual-tree fast Gauss transforms," *Advances in Neural Information Processing Systems (NIPS)*, vol. 18, pp. 747–754, 2006.

[5] C. Yang, R. Duraiswami, N. A. Gumerov, and L. Davis, "Improved fast Gauss transform and efficient kernel density estimation." in *Proc. of International Conference on Computer Vision (ICCV)*, vol. 1, 2003, pp. 464–471.

[6] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian kd-trees for fast high-dimensional filtering," in *ACM Transactions on Graphics (TOG)*, vol. 28, no. 3. ACM, 2009, p. 21.

[7] E. Dougherty, *Digital Image Processing Methods*. CRC Press, 1994.

[8] R. Deriche, "Fast algorithms for low-level vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 12, no. 1, pp. 78–87, 1990.

[9] D. Lang, M. Klaas, and N. de Freitas, "Empirical testing of fast kernel density estimation algorithms," University of British Columbia, Technical Report UBC TR-2005-03, 2005.

[10] P. Getreuer, "A survey of Gaussian convolution algorithms," *Image Process. On Line*, vol. 3, pp. 276–300, 2013.

[11] S. Yoshizawa and H. Yokota, "Fast L^1 Gaussian convolution via domain splitting," in *Proc. of IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 2908–2912.

[12] T. Ooura, *General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package*. www.kurims.kyoto-u.ac.jp/∼ooura/fft.html, 2006.

Real-time Video Stabilization using MEMS-sensors

Anastasiya Kornilova
Saint Petersburg
State University
kornilova.anastasiia@gmail.com

Iakov Kirilenko
Saint Petersburg
State University
y.kirilenko@spbu.ru

Natalia Zabelina
Saint Petersburg
State University
zabelina.nattaly@gmail.com

***Abstract*—This article describes our ongoing research on real-time digital video stabilization. It also explains and analyzes existing approaches to stabilization that use data from MEMS motion sensors. Authors propose to use the described methods for stabilizing the video that is transmitted to the mobile robot operator who controls the vehicle remotely, as well as increasing the precision of video-based navigation for subminiature autonomous models. This article also considers the main problems that came up during the experiments that were not resolved in the previous research papers. Authors offer possible solutions to these problems that would help improve quality of the work of existing algorithms.**

I. Introduction

Modern cameras' matrices allow to take high-quality pictures that are comparable to professional photographs. However, the quality of video that they are able to record leaves much to be desired and lately it has grown into a problem that needs to be resolved. If modern devices could improve quality of video recording in real time it would not only enable owners of smartphones and action cameras to stream more beautiful and visually appealing video, but would also solve more significant problems. For instance, in case of remotely controlled mobile robots and drones (quadcopters) that perform area monitoring, the low quality of video drastically decreases the precision of control and also leads to greater fatigue of the vehicle operator.

In most cases, you need to get rid of camera shake to solve the problem of poor video quality. It can be achieved either by fixing camera in one place (alternatively, by cancelling out its movement using specially designed mechanisms) or by transforming the frames digitally in such a way so that the video becomes jitterless.

If you choose the first option, you will need special external devices, such as SteadyCam, GyroStick, gimbal (for drones), or specially designed lenses and matrices similar to those available in professional cameras. This approach is not only extremely costly, but also not always applicable. For example, it is impossible to install an external stabilizer on smaller flying vehicles.

If you opt for the second way, or digital stabilization, you will face the challenge of camera motion estimation and image warping (Fig. 1). Video editing software developers have already advanced significantly in this area. Products like Adobe Premiere[1], Deshaker[2], Movavi[3] are all already able to stabilize videos digitally. Similar functionality is also available on YouTube that uses the algorithm proposed in the work [1]. The main disadvantage of these algorithms [2], [3], [4], [5], [6] is the amount of calculations needed to determine the camera motion. This makes this method inapplicable for real-time video stabilization. Besides that, these algorithms only use the data available in the images themselves, which makes them unreliable in case the shot has poor lighting or features large moving objects.

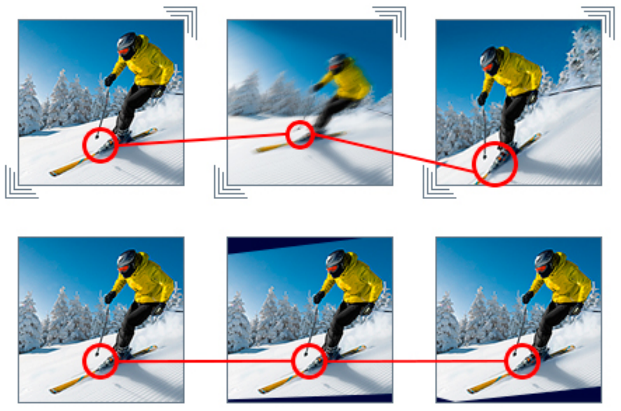


Figure 1. Image transformation for trajectory smoothing

Alternatively, you can estimate the camera motion during the recording by using the information from MEMS (Micro-ElectroMechanical Systems) motion sensors[4], including angular rate sensors (gyroscope), accelerometer and magnetometer. This method requires less processing power to determine camera positioning and, consequently, is more energy-efficient, which makes it suitable for real-time video stabilization. For instance, a common gyroscope consumes only 2-5 mW of power. At the same time, the CPU consumes several hundreds of milliwatts while analyzing frames.

This approach is applied more and more in recent years, as MEMS sensors are becoming widespread on different platforms, especially on smartphones. For instance, Google

[1]http://www.adobe.com/products/premiere.html
[2]http://www.guthspot.se/video/deshaker.html
[3]https://www.movavi.ru/
[4]https://en.wikipedia.org/wiki/Microelectromechanical_systems

Pixel, introduced in October 2016, completely lacks mechanical stabilization and uses only gyroscope-based stabilization algorithm. IPhone 7 also uses MEMS sensors for video stabilization but employs camera lenses and matrices for this purpose at the same time.

Mobile applications that offer similar functionality are just now coming up on the market and they are only able to perform video stabilization during post-processing. Some of the most prominent ones are: Instagram Hyperlapse[5], Microsoft Hyperlapse[6]. Gallus[7] is especially noteworthy, because, unlike others, it utilizes data from MEMS sensors.

This article considers different methods of real-time digital video stabilization that utilize MEMS sensors. Given that this research area is located at the junction of computer vision and digital signal processing, a lot of additional tasks arise, that are worth researching both separately and altogether. The main difficulties, when it comes to creating an application that allows to stabilize videos in real time, are the synchronization of frames and sensor data and the creation of a lightweight stabilization algorithm.

Authors review different existing algorithms and approaches as well as describe the problems that surfaced when these methods were implemented. During this research, we have encountered the following challenges: synchronization of frames and sensor reading, efficient frame transformation and increasing the accuracy of camera positioning. This article solves the found problems and offers more stable and universal implementation of the described algorithm.

In the second section of the article, we review the existing approaches to digital video stabilization that utilize MEMS-sensors, analyze whether these algorithms are suitable for use in real time and also list the mathematical models. In the third section, we describe the methods that improve positioning accuracy by using filters and combining readings from different sensors. In the fourth section, we analyze how to efficiently transform frames during camera rotation. In the fifth section, we consider the problem of synchronizing frames and sensor readings and use Android OS as an example. There we also review existing methods of automatic camera and sensor parameters calibration. In the sixth section, we list the main results of the ongoing research.

II. Video Stabilization

Video stabilization process can be divided into 3 independent stages:

1) estimating camera motion using MEMS sensors;
2) calculating the desired camera motion in accordance to some logic (for instance, trajectory smoothing);
3) transforming the frame to match camera motion to the desired one.

[5]https://hyperlapse.instagram.com/

[6]https://www.microsoft.com/en-us/store/p/hyperlapse-mobile/9wzdncrd1prw

[7]https://www.yafla.com/gallus/

In order to perform video stabilization in real time, we need to find a solution to each of the above listed tasks that would be satisfactory in terms of quality and performance.

The second stage is the most crucial. When smoothing trajectory, it's important to not only consider jitter as noise, but also to take into account that camera needs to move similarly to the way eye moves naturally. In the beginning of this section, we list the mathematical models and terms that are used and describe the existing algorithms. Then we analyze their advantages and disadvantages, and also propose various improvements.

Authors pay special attention to the two remaining stages, that can be improved significantly, yet still were not touched on in previous papers.

In this section, we suppose that all camera and sensor parameters are known, as well as that sensor readings and camera shots are synchronized in time. The abovementioned problems will be thoroughly discussed in the section dedicated to the parametrization of the stabilization system.

A. Mathematical models

Let's take a look at how frame is transformed when camera is rotated. We'll assume that x is the coordinates of a point on a projective plane, and X is the coordinates of a point in space (Fig. 2). Also, for each particular camera, let's assume that it has the matrix K with the following parameters: (o_x, o_y) is the optical center of the camera and f is its focal length. We'll get the following formulas for the projective transformation[7]:

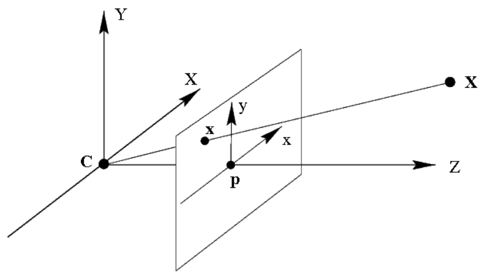


Figure 2. Projective transformation

$$x = KX$$

$$K^{-1} = \begin{pmatrix} 1 & 0 & -o_x \\ 0 & 1 & -o_y \\ 0 & 0 & f \end{pmatrix}$$

Let's fix the global coordinate system and assume, that in moment t the camera is rotated against it, using the rotation matrix $R(t)$[8] (Fig. 3). Then projective transformation will look this way:

$$x = KR(t)X$$

[8]https://en.wikipedia.org/wiki/Rotation_group_SO(3)

Let's assume, that x_i и x_j are both projections of the same point X in space, but they are located in frames i и j respectively, meaning:

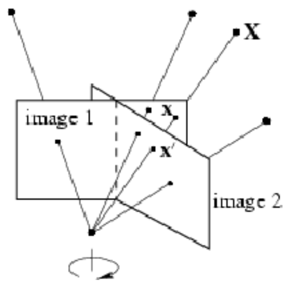


Figure 3. Location of a point in frames during camera rotation

$$x_i = KR(t_i)X$$

$$x_j = KR(t_j)X$$

By transforming these expressions, we establish the following connection between projections of the same point in different moments of time:

$$x_j = KR(t_j)R^T(t_i)K^{-1}x_i$$

Thus, let's define the matrix of image transformation between moments in time t_1 и t_2 as:

$$W(t_1, t_2) = KR(t_1)R^T(t_2)K^{-1}$$

$$x_j = W(t_j, t_i)x_i$$

We want to include an additional parameter to the above-described mathematical model of camera and its rotations. It's defined by the camera shutter and solves the problem of blurring when recording fast moving objects. Rolling shutter[9] is a visual distortion that happens, because when the shutter is released, each row of the frame is shot at a different moment in time (Fig. 4-5)[10].

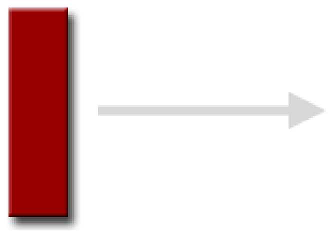


Figure 4. Object movement

[9]https://en.wikipedia.org/wiki/Rolling_shutter

[10]Images are taken from the website http://www.red.com/learn/red-101/global-rolling-shutter

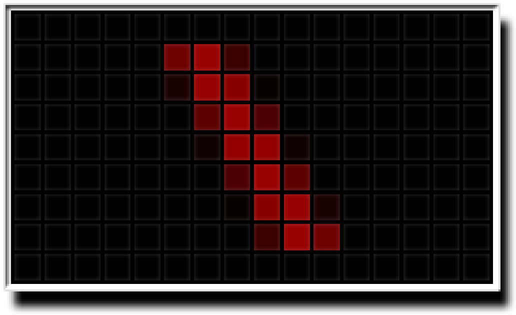


Figure 5. Rolling-shutter effect during capturing the moving object

When shutter scans the scene vertically, the moment in time at which each point of frame is shot, is directly dependent on the row it is located in. Thus, if we assume that i is the number of the frame and y is the row of that frame, then the moment at which it was shot can be calculated this way:

$$t(i, y) = t_i + t_s\frac{y}{h}$$

where t_i is the moment when frame number i was shot, t_s is the time it takes to shot a single frame, h is the height of the frame. This can be used to make the general model more precise, when calculating the image transformation matrix.

B. Stabilization algorithms

Among the solutions discussed in the scientific society, two are especially worth noting, and we will describe them in this section.

1) Algorithm with Gaussian filter

Algorithm described in the article [8] in 2011, is based on Gaussian filter[11]. Camera positioning is calculated by integrating the readings of a MEMS gyroscope for each frame. Then the sequence of camera movements is smoothed by utilizing the Gaussian filter (Fig. 6), and the frames are sequenced using the new motion model. Gaussian filter can be customized by changing the window size (how many discrete points it effects) and the size of the core (how strong the smoothing is). By altering these parameters one can either get rid of local jitter or significant movements.

The use of Gaussian filter is very effective during post-processing, but is not always applicable for real-time stabilization. During post-processing movement can be analyzed completely from start to finish, which allows to increase the size of the window of the filter and smooth the movement stronger. During real-time stabilization, processing buffer needs to include 10-15 frames, which results in a significant delay of 0,3-0,5 seconds.

The source code of the prototype was presented in Matlab, but the article states that algorithm was tested on an IPhone 4. During open realization, the algorithm features narrowed camera rotation parameters. Namely, only horizontal camera rotation is taken into account, which does not always reflect the movement of a shaking camera.

[11]https://en.wikipedia.org/wiki/Gaussian_filter

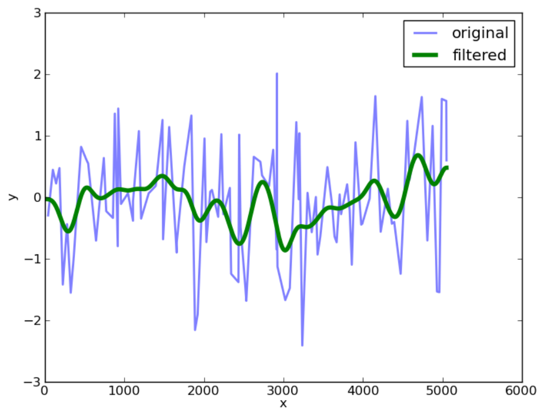


Figure 6. Trajectory smoothing using the Gaussian filter

2) Algorithm utilizing nonlinear filter

Algorithm described in the article [9] in 2014 utilizes a more complex nonlinear filter to smooth camera movement.

In the offered method, the definition of a virtual camera is given. Two concentric zones are selected on the frame – the inner region and the outer region (Fig. 7)[12]. Then the rectangle zone is selected in the inner region. Positioning of a virtual camera is determined by the position of this rectangle.

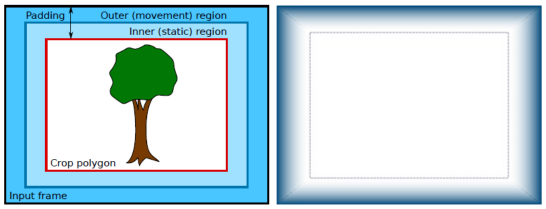


Figure 7. Inner and outer stabilization zones

For each new frame, a new position of the abovementioned rectangle is calculated. If it lies within the inner zone, the camera orientation remains the same. If any part of the rectangle lies outside the inner region then the virtual camera's angular velocity is updated by using spherical linear interpolation – slerp[13](spherical linear interpolation) to bring it closer the physical camera's velocity. Authors note that this algorithm works rather well, but when rectangle hits the edge of the inner zone sudden changes can be expected.

The article offers a way how to make this method suitable for real time video stabilization. If a buffer has k frames, than the camera is supposed to move during these frames with the same velocity it did before. If the rectangle crosses the inner zone, then the spherical interpolation is used to bring the virtual camera velocity closer to the velocity of the physical camera.

Besides significantly decreasing the buffer size, this method has one more advantage. It does not take into account the absolute positioning of the camera, as it only uses the velocity of the camera. Therefore, due to the absence of integration, the error is not accumulated.

Sadly, the authors of the article did not offer a repository with source code of the program, realizing this algorithm. Therefore, it was impossible to repeat the experiment at the time. We plan to realize this approach in the nearest future.

III. Determining the positioning

When we were constructing the above-described model, it was assumed that the sensor readings are continuous and accurate. In reality, however, as in all physical devices, MEMS sensors have noise. If the algorithm requires integrating the gyroscope readings, the error caused by the noise will only increase. To solve this problem we will combine the readings of two or more different MEMS sensors, for instance gyroscope and accelerometer. This will allow to eliminate significant errors. The following filters offer similar functionality:

1) Complementary filter[14];
2) Madgwick filter[10] – filter that utilizes the gradient descent and allows the use of magnetometer;
3) Mahony filter[11];
4) Extended Kalman filter – the most successful realization is presented in the work [12].

It is important to mention that the processing complexity of the offered algorithms needs to be minimized for real-time video stabilization. The algorithms are listed in the increasing order of complexity. It is worth noting, that the use of quaternions for estimating positioning and integrating is significantly less complex than other positioning methods like Euler angles or rotation matrices[13].

IV. Frame transfomation

After it was determined how much the frame positioning should change, projective transformation should be performed. Realization of the OpenCV library[15] offers this functionality via warpTransoform and perspectiveTransform functions. The first option performs projective transformation for the whole image, while the second one allows to determine the position of particular points on the frame after transformation.

Using the second function allows us to realize the following algorithm. We choose several points on the frame, a 10x10 grid, for instance. After that a projective transformation is performed for each point, and their new positions are calculated (Fig. 8). The values in the other spots are calculated using interpolation.

By varying the size of the grid, it is possible to find the balance between quality of the image after the rotation and speed of processing of the new frame. While experimenting with 1920x1080 frames, it was determined that the best results are achieved with 10x10 grids.

[12]Imageistakenfromthearticle\cite{nvidiaStab}
[13]https://en.wikipedia.org/wiki/Slerp

[14]https://en.wikipedia.org/wiki/Alpha_beta_filter
[15]http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html

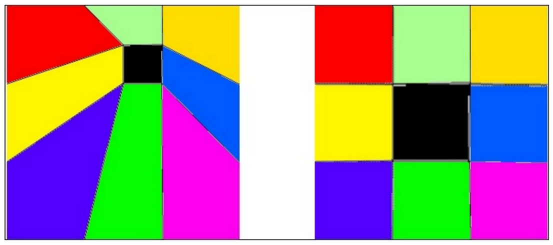


Figure 8. Image warping

V. Camera calibrations and synchronization

Camera model and the stabilization algorithms, described above, are based on certain assumptions that are not always true in reality. First, it is assumed that sensor readings are a continuous function and are synchronized with frames. Second, we assume that all the necessary parameters for the mathematical model, such as: optical center, focal length and shutter release time are known.

In this section we describe these issues in more detail and offer different solution to the problems.

A. Calibrating the unknown camera and sensor parameters

In order for stabilization algorithms to work correctly, we need to have detailed information about camera's and MEMS sensors' parameters. Namely, the optical center, focal length, location of the MEMS sensor coordinate axes in relation to the camera's coordinate axes and shutter release time (rolling shutter). Assuming all pixels are square, we'll set the optical center at $(0, 0)$.

In case of all sensors, a gyroscope in particular, the main unknown parameter is the bias[16] – almost constant skew of angular velocities against the exact measurements. Smartphones sensors are calibrated automatically, while in case of some embedded systems, you need to monitor this parameter closely, as the bias can result in error during integration. To determine the bias, we need to find the mean deviation of angular velocities against the null, when the camera is stable.

The calibration and synchronization problems are solved in the article [14], where the process of online calibration using the extended Kalman filter is described in full detail. Also, in the article [15] the minimization method including determining of the cost function is offered to calibrate the parameters listed above. The full review of camera parameters calibration methods is available in the article[16].

Currently, authors select camera parameters manually for the models used to test algorithms. Automatic calibrations will be realized only after successful experiments with the algorithms.

B. Synchronization of a camera and sensors

First, it is important to understand that MEMS sensor readings are discrete. Therefore, even if you know the exact time each frame was taken, it would be impossible to determine the current positioning of the camera. However, since signal's frequency of the MEMS sensor is between 100 and 200 Hz and the frame rate is 30 fps, we can use simple interpolation to get a relatively accurate estimation.

Unlike embedded systems, that offer hardware synchronization of frames and MEMS sensor reading, operating systems of smartphones sometimes do not offer this functionality. Authors encountered this problem on Android when prototyping the application for simultaneous recording of video and data from sensors.

It turned out, that the main API of the camera[17], available on each phone does not provide the event scheme for processing single frames. Therefore it was impossible to use software to determine the place of each frame in the time series of sensor readings (Fig. 8)[18]. The possible solution to this problem is using the mathematical methods to match two time series with different degrees of discretization: frequent – sensor reading and rare – video frames. The use of displacement of features[19] as metric is suggested.

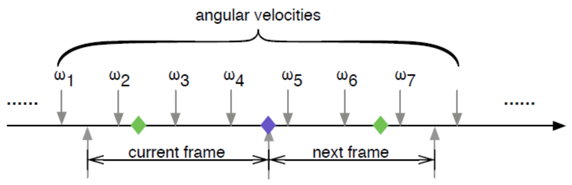


Figure 9. Matching the time series of frames and gyroscope

Starting with level 21 Android API, a new API for Camera2[20] was introduced. It features the event driven programming that would allow to determine the taking of a frame by using the event handler OnImageAvailableListener[21]. Even if this improvement can't be used to determine the exact timestamp of a frame, it will help to estimate the place of the frame on the time series of sensor readings. Therefore, this approximation can be used for realizing the mathematical method for matching series.

VI. Current results

Currently, authors have implemented the prototype of the algorithm utilizing the Gaussian filter on Python, that cover the model of 3-dimensional camera rotation. Provided the synchronized sensor readings and frames, as well as intrinsic camera parameters, this algorithm shows great results during post-processing.

Synchronization of sensor readings and camera is performed by an application, described in the corresponding section. Based on this, we plan to execute this algorithm in real-time

[16]http://www.vectornav.com/support/library/gyroscope

[17]https://developer.android.com/reference/android/hardware/Camera.html

[18]Image taken from the article [14]

[19]https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)

[20]https://developer.android.com/reference/android/hardware/camera2/package-summary.html

[21]https://developer.android.com/reference/android/media/ImageReader.html

mode in the nearest future. To decrease latency we will use the optimal filters, that are described in the section dedicated to them, as well as piece-by-piece frame transformation.

To make the software video stabilization module cross-platform, we plan to test the suggested methods of real-time calibration of intrinsic camera parameters and implement them.

Conclusion

At this moment, there are many different approaches to digital video stabilization, but not all of them require too much processing power to be used in real time use. Methods utilizing MEMS sensors are worth noting as they allow to save processing resources. Scientific community offers several stabilization algorithms utilizing these sensors. They show great results during post-processing and several prototypes for real-time processing are available.

Despite the possibilities and the need for real-time digital stabilization, its implementation is hard from a technical standpoint, because the video sensor and MEMS sensors need to be coordinated. Besides that, a lot of work still needs to be done to optimize these algorithms for work in real-time.

Many additional challenges and problems described by the authors, show that there is a lot of room for improvement in existing solutions, namely in the way algorithms work. All algorithms that we studied employ a quite primitive mathematical model, which makes it viable to continue research in this area using more advanced mathematics. Authors set their next goal as using the work they have already done to build a full-fledged software module for real-time digital video stabilization and increase its ability to function on different platforms.

VII. Acknowledgment

Funding for this work was provided by JetBrains Research.

References

[1] M. Grundmann, V. Kwatra, and I. Essa, *Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths*. 2011.

[2] Y. Matsushita, E. Ofek, W. Ge, X. Tang, and H. Shum, *Full-frame video stabilization with motion inpainting*. 2006.

[3] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala, *Subspace video stabilization*. 2011.

[4] M. Grundmann, V. Kwatra, and I. Essa, *Auto-directed video stabilization with robust l1 optimal camera paths. In: Proceedings of CVPR*. 2011.

[5] Y. Wang, F. Liu, P. Hsu, and T. Lee, *Spatially and temporally optimized video stabilization*. 2012.

[6] S. Liu, L. Yuan, P. Tan, and J. Sun, *Bundled camera paths for video stabilization*. 2013.

[7] R. Szeliski, *Computer Vision: Algorithms and Applications*. 2010.

[8] A. Karpenko, D. Jacobs, and J. Baek, *Digital Video Stabilization and Rolling Shutter Correction using Gyroscopes*. 2011.

[9] S. Bell, A. Troccoli, and K. Pulli, *A Non-Linear Filter for Gyroscope-Based Video Stabilization*. 2014.

[10] S. O. Madgwick̆, *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. 2010.

[11] R. Mahony, T. Hamel, and J.-M. Pflimliń, *Complementary filter design on the special orthogonal group $SO(3)$*. 2005.

[12] R. Zhua, D. Sunb, Z. Zhoua, and D. Wangá, *A linear fusion algorithm for attitude determination using low cost MEMS-based sensors*. 2007.

[13] J. Diebel, *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. 2006.

[14] C. Jia and B. L. Evans, *Online Calibration and Synchronization of Cellphone Camera and Gyroscope*. 2013.

[15] H. Ovren and P.-E. Forsseń, *Gyroscope-based Video Stabilisation With Auto-Calibration*. 2013.

[16] W. Qi, F. Li, and L. Zhenzhong, *Review on Camera Calibration*. 2010.

[17] J. Howse, S. Puttemans, Q. Hua, and U. Sinha, *OpenCV 3 Blueprints*. 2016.

[18] Invensense, *MPU-6000 and MPU-6050 Product Specification*. 2012.

Type-2 Fuzzy Rule-Based Model of Urban Metro Positioning Service

Albina R. Gimaletdinova, Konstantin Y. Degtiarev (IEEE Member)
School of Software Engineering
Faculty of Computer Science
National Research University Higher School of Economics
3 Kochnovsky Proezd, 125319, Moscow, Russian Federation
argimaletdinova@edu.hse.ru, kdegtiarev@hse.ru

Abstract — **Despite the large body of existing knowledge on navigational services, there has been an important issue relative to positioning accuracy. The paper discusses a possible solution to comparison problem, which is linked to the determination of the closeness to destination metro station through finding a difference between user's current coordinates and fixed point coordinates. With this end in view, fuzzy logic approach is used to develop Routes Recommender System that utilizes linguistic variables to express the vague and uncertain term 'closeness to…'. The paper provides detailed explanation of each variable considered in fuzzy inference system (FIS), set of fuzzy rules in line with graphical representation of system's output. Based on Mamdani model, we propose test cases to check maintainability of the model.**

Keywords — positioning service; mobile applications; fuzzy modeling; GPS; WiFi; Cellular Networks; public transport; interval type-2 fuzzy sets (IT2FS); fuzzy inference system; fuzzy matching of coordinates; uncertainty.

I. Introduction

Over the past decade positioning techniques have become common in almost all branches of industry. In particular, nowadays vast majority of phone models are provided with GPS-module that can be enabled in different cases. Positioning feature is rather common to mobile applications supporting navigational services, and the latter can be used by people in urban transport. The purpose of the paper is to exploit the potentialities of fuzzy logic regarding recommender system with the navigational service. Such service may solve the problem of frequently encountered disorientation of passengers in unfamiliar terrain and allow to pave routes between stations of interest (case of urban transportation system). The potential application may notify a passenger about forthcoming arrival, when he/she is situated closely to the end station. The main purpose in the present context is to determine a deviation between current and end points (stations). Consequently, it leads to the serious problem, since we cannot precisely assert whether a user is close to the end station or not. It occurs because there is a need to estimate the smallest difference (delta) between current and end-point coordinates and then set rule(-s) to classify user`s location.

The issue of applying fuzzy logic to positioning, tracking and transportation attracts attention of researchers. Selected publications have focused on indoor positioning. For example, Chen C.-Y., Yung J., et al. [1] studied indoor positioning technique based on received signal strength and fuzzy approach; they showed experimentally that such method has better performance as compared to geometric triangulation method [2] – actually, the same objective was pursued in the research by Teuber A. and Eisfelller B. [2]. The fuzzy system to control train automatic stop, with the emphasis on stop accuracy, was developed by Yasunobu S., Miyamoto S. and Ihara H. in [3]. It is evident that the practical application of fuzzy logic to positioning or transportation subject matter cannot be considered as exclusive one, however, the issue of positioning in metro should be studied in details.

As it was mentioned above, the study is devoted to indoor positioning within the metro transportation system. We make an attempt to develop a fuzzy model of metro stops allowing to send timely destination notifications to passenger. It is clear that we do not know exact minimum and maximum distances between stations or the moment when the application should send a reminder. Uncertainty has many faces and forms of manifestation. As stated by George J. Klir and Mark Wierman, "uncertainty involved in any problem-solving situation is a result of some information deficiency; … information may be incomplete, fragmentary, not fully reliable, vague, contradictory, or deficient in some other way" [4]. Hence, when we do not know or cannot obtain exact values/parameters of some phenomena (e.g. distances between points, the location of some moment on a time scale), we need to deviate from type-1 fuzzy sets as a general framework to handle *vagueness* (for more information see seminal papers "Fuzzy Sets" (1965) and "The Concept of a Linguistic Variable and Its Application to Approximate Reasoning – I" (1975) by L. Zadeh) to more general type-2 fuzzy sets that allow to reflect the uncertainty in adequate, more thorough manner, or, put it precisely, to model it. In the work interval type-2 fuzzy sets (IT2FS) are used; to ensure computational efficiency, the preference is given mainly to piecewise linear functions (trapezoidal shape) as upper and lower membership functions of IT2FS.

The rest of the paper is organized as follows: the second section explains the main problem that the paper is devoted to. Section 3 provides definition of linguistic variable (LV) and describes those variables and their linguistic values represented in the form of type-2 membership functions that are used in the inference process (Mamdani's fuzzy model); explanations on domains (universal sets) for each variable are also adduced in this section. The following section 4 makes emphasis on fuzzy rules that serve as a basis for fuzzy system (model developed), covers short comments on type-reduction defuzzification methods used in the study; results of experiments with the system under different values of input variables are presented in both tabular and graphical forms. Section V of the paper concludes explicitly mentioning the ways of further elaborating upon the subject.

II. Problem Definition and General Comments

One of the main issues we have to deal with is to find a user's position. Current position obtained should be compared with fixed station's coordinates (e.g. end-point of the route or interchange point to other line) – it will allow to say where is a user now. If he/she is close to one of the points, the application should signal to him about it, thus the understanding and definition of the word "close" becomes essential. The factor of closeness is treated unequally by different people, and a nearby object for one person can be far away for another one. It means that estimation of closeness relates to certain difficulties and, as a consequence, we cannot associate crisp numbers as a basis for possible values of the variable "close". Therefore, the only way to describe closeness at a first approximation is to set a numerical interval of its possible values and to use it at further processing steps.

We may assume that in the beginning the application gets start and end points of the route (input data), then it ensures passenger tracking using one of the positioning technologies (GPS, WiFi, Cellular Networks) and compares his/her current location with the one of key points. According to [5] and practical everyday experience, GPS has poor accuracy indoors, including metro, therefore, we do not consider GPS positioning accuracy to calculations shown in Table 1. As already mentioned before, we will use numeric interval to represent difference between fixed and current coordinates.

TABLE I. Approximate Accuracy for Different Positional Techniques ([6, 7])

№	Technique	Min accuracy (m)	Medium accuracy (m)	Maximum accuracy (m)
1	GPS	2	11	20
2	WiFi	10	80	150
3	Cellular	100	800	1500
	Average	37.3	297	557
	Average for №2 and №3*	55	440	825

* The last row is calculated without GPS characteristics (signal in metro is bad)

Received data concerning current position can be inaccurate, because positioning techniques used in the phone do not guarantee 'ideal' precision of geographical coordinates supplied because of various objective reasons (e.g. tracking indoors or underground, bad quality of signal from provider, etc.). Thus, it makes sense to emphasize another overt source of *fuzziness*, which relates to fuzzy (vague) matching of coordinates – latitude and longitude indicators will be analyzed separately.

III. Fuzzy Logic Model: Definition of Lingustic Variables and Their Values

Firstly, we should select input-output variables for fuzzy system and provide necessary explanations. All significant internal and external factors, in which uncertainty shows itself, must be analyzed; this is an important stage in development of the model. Internal factors signify that certain issues depend solely on application itself (its realization), and some tuning steps can lead to better results. On the contrary, external factors indicate that there is obvious reality that is not dependent on realization per se – these are the factors that most of people are familiar with, viz. bad quality of signal from provider, poor WiFi coverage, etc.

In order to explain internal factors it is necessary to detect variables at the level of passenger's tracking; the central operation here is obtaining a current position. Once it is done the difference between fixed point coordinates and current point must be determined – we call this difference (i.e. variable) "*Difference between fixed and current points (delta)*".

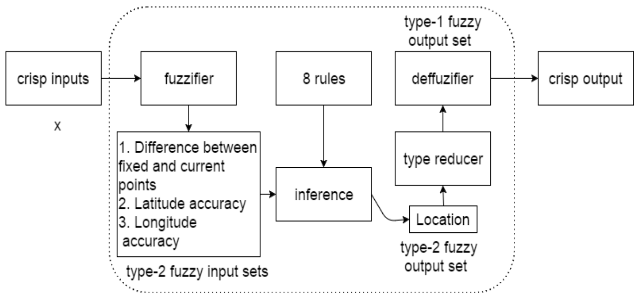


Fig. 1. Fuzzy model with names of linguistic variables in use

If we use WiFi and Cellular Networks, it means that the application is going to get coordinates with different accuracy, even at the same place without any movements. We have to admit that the factor of fuzziness definitely becomes apparent in the problem of coordinate matching, and the accuracy will depend on chosen positioning method (see Table 1). We will combine two techniques mentioned above, and because of that Table 1 contains cells with calculated average accuracy. In the paper we take into account possible accuracy of latitude and longitude – let's name these variables as "*Latitude accuracy*" and "*Longitude accuracy*". The output of fuzzy system will represent position respective to metro station. All these variables as inputs and output of rule-based system to be used are shown in Fig. 1.

Variables mentioned above in the text are *linguistic variables*, i.e. their values are words (or, phrases) of natural language; formally, these values are fuzzy sets, and they are represented by membership functions. In general, a linguistic variable is defined as a tuple $\langle L_v, \mathrm{T}(L_v), \mathrm{U}, \mathrm{G}, \mathrm{M}\rangle$, where L_v is the name of the variable (e.g. $L_v \equiv$ "*Latitude accuracy*"), $\mathrm{T}(L_v)$ is the set of labels of variable's L_v linguistic values $l_1,..,l_n$ (term-set of L_v; e.g. $l_i \equiv$ 'insignificant', etc.). The names (labels) are generated using syntactic rule G, the meaning

$\mathrm{M}(l_i)$ is associated with each value $l_i, i=\overline{1,n}$, from $\mathrm{T}(L_v)$; $\mathrm{M}(l_i)$ is a fuzzy set (respective membership function) defined on a universe of discourse (domain) U. The latter must be defined for all input and output variables introduced earlier. Thus, every variable is characterized by its own set of acceptable values and membership functions for each such value $l_i, i=\overline{1,n}$.

A. Linguistic variable "delta" and its values (terms)

Earlier we were talking about the difference between fixed and current points (so-called *delta*). What does it really mean? The value that expresses the difference falls into the interval $[0,a]$, where real-valued $a>0$ (deviation is analyzed in absolute magnitude); its left bound (0) means that passenger's coordinates are similar (better to say, close) to some fixed point. We assume that the application should notify a passenger outright before a given destination, when he/she is at the station that precedes terminal station of the route, or at some later moment. Consequently, we consider the average distance between two stations, and a passenger should have enough time to alight from the railway (metro) carriage without effort.

To calculate the biggest difference between coordinates, we should estimate the average distance between any two stations in the metro and double it, because at this moment it will be not an urgent question to notify a passenger about the arrival as he/she still has to go two or more stations more. Following [8, 9], the mean distance between stations in Moscow metro is equal approximately to 1,780 meters. Hence, *a* value signifies the biggest possible difference, i.e. 40,075,000 meters (the length of Earth's equator) = 360° (circle grade measure)

$$1,780\times 2=\frac{1,780\cdot 2\cdot 360^{\circ}}{40,075,000}\approx 0.032^{\circ} \qquad (1)$$

Therefore, values of *delta* are limited to the interval $[0, 0.032]$ (in degrees) that relates to domain (universal set) U, over which linguistic variable $L_v^{(1)}\equiv "delta"$ is defined. Yet, why do we talk about linguistic variable in that case? In the everyday life people prefer to use words or phrases of the natural language as a habitual terms (values) for description of phenomena they are dealing with in their diverse activities. In case of *delta* variable such attached to it terms as 'big', 'small', etc., on one hand, form a solid ground for communication within the professional medium allowing almost uniform apprehension of the meaning of these values. On the other hand, their inherent uncertainty has to be adequately modeled when used in computational methods. In particular, we may introduce 2 linguistic terms 'small' and 'bigger' (difference between coordinates) as applied to the variable *delta*. Since type-1 membership functions (T1MF) are precise, i.e. the degree of belongingness $\mu(x)$ of each generic element x to corresponding fuzzy set is a crisp number, T1MF cannot represent the typical uncertainty intrinsic to estimates $\mu(x)$ (tilde sign emphasizes the fact that these degrees are not reducible to ordinary numbers). Linguistic values can be represented in the form of interval type-2 fuzzy sets (IT2FS); the latter are characterized by Lower (L) and Upper (U) membership functions that bound the area called footprint of uncertainty (FOU). The shape of this region allows to express the uncertainty in $\mu(x)$ estimates obtained, providing "additional degrees of freedom … to handle MF uncertainties" [10]. For each $x\in\mathrm{U}$, where U is a universe of discourse under consideration, all points in the range $\left[\mu^{(L)}(x),\mu^{(U)}(x)\right]$ may have equal unitary weights, i.e. secondary membership function defined on this interval is constant one. For practical reasons, such IT2FS seem to be convenient enough, accurate from the standpoint of giving proper weigh to uncertainty represented and most easily understood by stakeholders. Henceforth, just this kind of T2FS is used in the model with the direction of attention toward piecewise-linear type (trapezoidal case) of L and U membership functions.

Firstly, it is needed to define trapezoidal MF in terms of L and U functions' parameters for each linguistic value (term) – all calculations are done in accordance with (1). We assume that $L_v^{(1)}\equiv "delta"$ is associated with the term-set $\mathrm{T}(L_v^{(1)})=\{l_1,l_2\}=\{\text{'small difference'},\text{'bigger difference'}\}$ with 2 elements (Fig.2). The upper function (U) for the term 'small difference' of the variable *delta* can be characterized by parameter's set A(0,0), B(0,1), C(0.008,1) and D(0.016,0); the x-coordinate of the point C is the average of x-coordinates of parameters B ($\mathrm{B_x}$) and D ($\mathrm{D_x}$), the latter is the distance between any 2 stations. In much the same way, for the lower function (L) corresponding parameters are A(0,0), B(0,1), C(0.004,1), D(0.008,0).

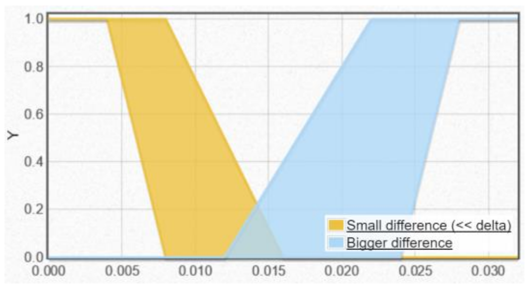


Fig. 2. Difference between fixed and current points (values of *"delta"*)

The 4-tuple of the upper function (U) that represents the linguistic term 'bigger difference' of *delta* is A(0.012,0), B(0.022,1), C(0.032,1) and D(0.032,0), where $\mathrm{A_x}=\left(\mathrm{D_x^{(L'small')}}+\mathrm{D_x^{(U'small')}}\right)/2=0.012^{\circ}$, both x-coordinate $\mathrm{C_x}$ and $\mathrm{D_x}$ are set to maximum difference 0.032 (1), the value of $\mathrm{B_x}$ is calculated as a mean of two neighboring points $\left(\mathrm{A_x}+\mathrm{C_x}\right)/2=0.022^{\circ}$. It's worth noting that not-yet-application will receive latitude and longitude coordinates as input data, so values are bound to degrees, but not meters. For the lower function (L) set of its parameters takes the form A(0.024,0), B(0.028,1), C(0.032,1), D(0.032,0); again, the value that relates to maximum difference appears here, the $\mathrm{B_x}$ value is obtained much as shown above, and $\mathrm{A_x}$ equals to the sum of $\mathrm{A_x^{(U'bigger')}}$

and the width of the left tail constituting an approximate half of the distance between stations (890 m) converted to degrees.

B. Linguistic variable "latitude/longitude accuracy" and its values (terms)

$L_v^{(2)} \equiv$"*latitude/longitude accuracy*" is the next variable to consider. As the telephone receives positioning information due to a correction to be made for the accuracy, it must be taken into account in calculation of difference between fixed and current points. The variable $L_v^{(2)}$ is defined on the interval $[0,b]$, where $b>0$ is the maximum of average accuracy as shown in the last row of Table 1. The not-yet-application doesn't allow to use GPS in metro, so we consider combined usage of WiFi and Cellular Networks. All calculations shown below are based on values summarized in Table 1, and they are performed in line with (1), i.e.

$$\begin{gathered}440\text{ m}\approx 0.00395^\circ;\ 825\text{ m}\approx 0.007^\circ\\ 55\text{ m}\approx 0.00049^\circ;\text{ mean of min and medium}\\ (55+440\text{ m})\approx 0.00444^\circ\end{gathered} \tag{2}$$

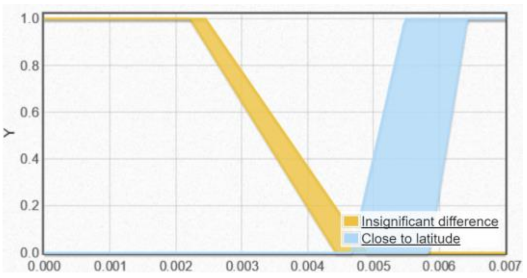


Fig. 3. Values of variables "*latitude/longitude accuracy*" (same graph)

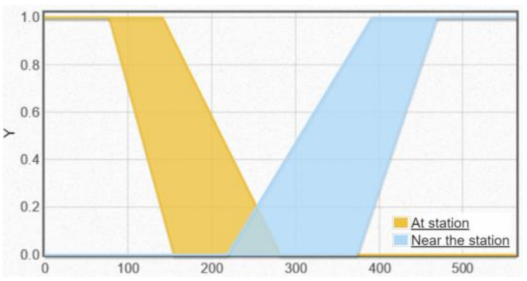


Fig. 4. Two values of the linguistic variable "*location*"

Thus, the universe, on which variable $L_v^{(2)}$ is defined, results in $\text{U}=[0,0.007]$ (2). The upper function (U) for the term 'insignificant difference' of the variable $L_v^{(2)}$ can be characterized by parameter's set A(0,0), B(0,1), C(0.00245,1) and D(0.0049,0); C_x is calculated as the arithmetic mean of B_x and D_x, which is the minimal average accuracy shown in Table 1. As for the lower function (same linguistic term is considered), the values of its parameters are A(0,0), B(0,0), C(0.00222,1) and D(0.00444,0). First two parameters reflect perfect accuracy at the position; C_x is obtained as before, while D_x value corresponds to (2). Linguistic values 'close to latitude/longitude' should be viewed separately, because for each component of coordinate's pair factor of inaccuracy (its measurement) sounds alike, but still differently. Their presence leads to more stable model (Fig. 1) and helps to improve the results attained. The upper function (U) is determined by parameters A(0.00467,0), B(0.00548,1), C(0.007,1) and D(0.007,0); $\text{A}_\text{x}=\left(\text{D}_\text{x}^{(\text{L'insig.diff'})}+\text{D}_\text{x}^{(\text{U'insig.diff'})}\right)/2=0.00467^\circ$, B_x is an arithmetic mean of A_x and C_x. For the lower function (L) parameters are specified as follows: A(0.00584,0), B(0.00642,1), C(0.007,1) and D(0.007,0); $\text{A}_\text{x}=\left(\text{A}_\text{x}^{(\text{U'close to latitude'})}+0.007^\circ\right)/2=0.00584^\circ$, B_x is calculated much as it is done in the case of upper function (U), both C_x and D_x are equated with the value of 0.007° that stands for minimum accuracy (or, maximum inaccuracy) – corresponding values are shown in Fig. 3. In the case concerned, only non-negative values of accuracy are considered; if calculations lead to negative result, we use its modulus.

C. Linguistic variable "location" and its values (terms)

The variable $L_v^{(3)}\equiv$"*location*" is the next matter under discussion – actually, it expresses the location, as it arises from variable's name, of a passenger due to indications related to previously mentioned variables. The variable is represented graphically in Fig. 4. We introduce two values (fuzzy sets) of $L_v^{(3)}$, namely, they are 'at station', i.e. main region that must be reached to notify a user about the arrival, and 'near the station'. The standard length of Moscow metro' platform is appr. 155 meters (8 train carriages), the longest station is "Vorobyovy Gory" – its length is about 282 meters [11]. The universe of discourse U the variable $L_v^{(3)}$ is defined on can be denoted as $[0,c]$, where the right bound *c* equals to the double length of the longest platform in the metro. For the upper function (U) as a constituent of IT2MF representing value 'at station', we set the following parameters: A(0,0), B(0,0), C(141,1) and D(282,0), where C_x is a half of the longest station (282 m) in the Moscow's metro. The parameters of the lower function (L) of IT2MF are A(0,0), B(0,0), C(77.5,1) and D(155,0) with C_x calculated as the arithmetic mean of B_x and D_x coordinates. We suggest to model the linguistic value 'near the station' with the IT2MF, whose upper function (U) is characterized by A(218.5,0), B(391.25,1), C(564,1) and D(564,0); the value of A_x is obtained as $\left(\text{D}_\text{x}^{(\text{L'at station'})}+\text{D}_\text{x}^{(\text{U'at station'})}\right)/2=218.5$ (in meters), B_x is the mean of A_x and C_x x-coordinates, both C_x and D_x are equal to 564 meters (double length of the longest platform). Similarly, parameters of the lower function (L) are A(373.5,0), B(468.75,1), C(564,1) and D(564,0), where A_x (x-coordinate of the first parameter) equals to $\text{A}_\text{x}^{(\text{U'at station'})}+155=373.5$ that takes into account the length of the standard metro train, i.e. the latter will have direct influence on the spread of the left tail of the membership function. As before, coordinate B_x is the average of A_x and C_x (468.75 meters), and non-negative values are considered.

Rather detailed description of linguistic variables and their values is important for deeper understanding of fuzzy logic system (its model), the use of interval type-2 membership functions to represent uncertainty inherent in verbal values introduced and with the regard for specific character of possible implementation of the system in the code. To a large extent, the definition of a very small number of linguistic variables' values pursues two plain objects – namely, (1) to obtain the initial "non-overloaded" (in terms of number of values and fuzzy rules) variant of the system to perform experiments with and to lay a ground for further analysis, tuning parameters and rule base, revealing drawbacks, etc., and (2) to examine the general idea of using type-2 fuzzy sets in recommendation services that are actively advancing as it applies to enormous market of mobile devices.

IV. Rules of the Fuzzy Model (Inference System) and Experiments Conducted

The core of the fuzzy inference system (FIS) as shown in Fig. 1 is a set of linguistic values represented in the form of fuzzy sets, If-Then rules having a generic form ″<u>If</u> {*antecedent*} <u>Then</u> {*consequent*}″ and fuzzy reasoning scheme; the latter just operate on a given rules along with specified inputs to derive system's outputs or conclusions. The experts' understanding of the phenomenon under study and their knowledge of the domain field provide a basis for formation of the primary version of rule-base, in which linguistic variables $L_v^{(1)} \equiv$ "*latitude/longitude accuracy*" and $L_v^{(2)} \equiv$ "*delta*" are used in antecedent part of fuzzy rules (input of the system), whereas $L_v^{(3)} \equiv$ "*location*" operates as system's output (its terms form consequent part of rules). The evident transparency of the rule-base in general is substantiated here by a specific fact of simplicity and lucidity of both linguistic values submitted for consideration and existing relations between them. To the opinion of authors, such situation can be viewed as an advantage in terms of efforts needed to design the rule-base. However, it does not mean that the subsequent fine-tuning of rules as well as values of variables will not be needed – most likely, this stage is unavoidable in practice regardless of the system at hand. At the moment, the rules can be represented in the following form:

Rule 1	<u>If</u> *delta* is ′small difference′ and *latitude accuracy* is ′insignificant difference′ **and** *longitude accuracy* is ′insignificant difference′ <u>Then</u> *location* is ′at station′
Rule 2	<u>If</u> *delta* is ′small difference′ **and** *latitude accuracy* is ′close to latitude′ **and** *longitude accuracy* is ′insignificant difference′ <u>Then</u> *location* is ′near the station′
Rule 3	<u>If</u> *delta* is ′small difference′ **and** *latitude accuracy* is ′insignificant difference′ **and** *longitude accuracy* is ′close to longitude′ <u>Then</u> *location* is ′near the station′
Rule 4	<u>If</u> *delta* is ′small difference′ **and** *latitude accuracy* is ′close to latitude′ **and** *longitude accuracy* is ′close to longitude′ <u>Then</u> *location* is ′near the station′
Rule 5	<u>If</u> *delta* is ′bigger difference′ **and** *latitude accuracy* is ′insignificant difference′ **and** *longitude accuracy* is ′insignificant difference′ <u>Then</u> *location* is ′near the station′
Rule 6	<u>If</u> *delta* is ′bigger difference′ **and** *latitude accuracy* is ′close to latitude′ **and** *longitude accuracy* is ′insignificant difference′ <u>Then</u> *location* is ′near the station′
Rule 7	<u>If</u> *delta* is ′bigger difference′ **and** *latitude accuracy* is ′insignificant difference′ **and** *longitude accuracy* is ′close to longitude′ <u>Then</u> *location* is ′near the station′
Rule 8	<u>If</u> *delta* is ′bigger difference′ **and** *latitude accuracy* is ′close to latitude′ **and** *longitude accuracy* is ′close to longitude′ <u>Then</u> *location* is ′near the station′

A. Test 1 (difference between fixed and current points (delta))

The first carried out experiment is related to checking the difference between fixed and current points (i.e. linguistic variable ″delta″) under the constant latitude/longitude accuracies equal to 0.00074 (step of delta's change is taken as 0.0032, number of steps equals to 10). IT2MF is an assortment of type-1 membership functions embedded between upper (U) and lower (L) functions. Each of these embedded functions (type-1) can be defuzzified, viz. converted to crisp number that represents generically corresponding fuzzy set (its membership function). The most commonly used method of defuzzification is called centroid [10]. The processing of type-2

TABLE II. Centroid Type Reduction Defuzzification

№	Difference between fixed and current points (delta)	Location
1	0.0032	84.398
2	0.0064	94.312
3	0.0096	139.576
4	0.0128	282.000
5	0.016	392.995
6	0.0192	392.995
7	0.0224	392.995
8	0.0256	448.347
9	0.0288	460.487
10	0.032	460.487

fuzzy systems provides for the use of type reduction procedure (TRp) that can be seen as an expanded form of type-1 defuzzification resorting to Extension principle [12]. Each of rules **Rule i**, $i = \overline{1,8}$, "fires" and leads to obtaining output type-2 fuzzy set under a given input data. The union of these output sets and calculation of the centroid of resultant set is the essence of the centroid type reduction. Both theoretical framework and development of type reduction's use in type-2 fuzzy systems were presented in publications by Karnik N.N. and Mendel J.M. [13, 14]. As applied to IT2FS (secondary membership function in that case is constant), TRp becomes simpler in comparison with generalized type-2 sets – the results of experiment (see the data above) using centroid type reduction defuzzification as summarized in Table II.

TABLE III. CENTER-OF-SETS TYPE REDUCTION DEFUZZIFICATION

№	Difference between fixed and current points (delta)	Location
1	0.0032	84.398
2	0.0064	84.398
3	0.0096	84.398
4	0.0128	275.180
5	0.016	460.487
6	0.0192	460.487
7	0.0224	460.487
8	0.0256	460.487
9	0.0288	460.487
10	0.032	460.487

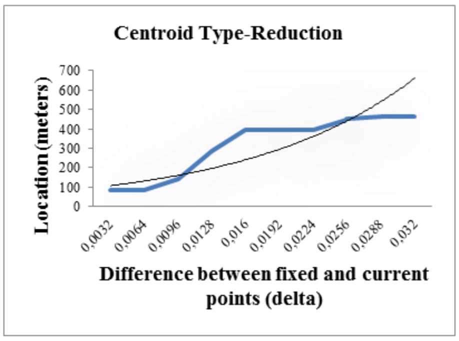


Fig. 5. Centroid type reduction method for *"delta"* variable

On the other hand, another TRp called center-of-sets type reducing approach (there is a family of defuzzification methods proposed up to now) can be used to substitute the consequent parts of rule-base by singletons at the centroid of corresponding fuzzy sets (Then-part of rules). Subsequent step is connected with obtaining the centroid of type-1 fuzzy set constituted by aforementioned singletons [10]. Calculated values that refer to test data (section IV, item's *A* preamble) are accumulated in Table III.

It can be noticed that for a particular set of test data centroid TRp demonstrates better (more smooth) approximation of the moderately growing exponential trend. Relative angularity (in Fig.5 it is not so strongly pronounced in comparison with Fig.6 case) relates to the use of piecewise linear (trapezoidal) functions representing fuzzy sets, certain (potential) drawbacks ascribed to rule-base design issues and small number of linguistic terms defined for each variable under consideration. However, even under these circumstances, results of centroid TRp indicate that it is more sensitive to accuracy changes (fine-tuning) than the second TRp. The second graph (Fig.6) visualizes marked broken line consisting of 2 constant levels, and one of those is rather lengthy. To a variable degree, both lines are increasing, and centroid TRp is preferable, since it takes into account specificity of all functions' values.

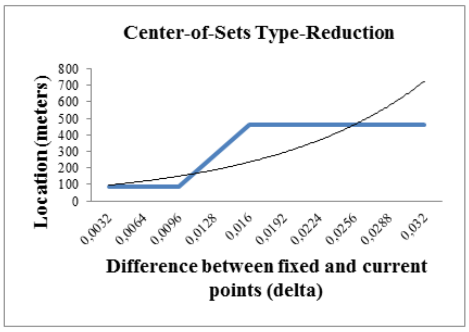


Fig. 6. Center-of-sets type reduction method for *"delta"* variable

B. *Test 2 (latitude/longitude accuracy)*

The second test relates to checking the latitude/longitude accuracy under constant difference between fixed and current points (*delta*) equals to 0.0032 (longitude accuracy is 0.00074 *OR* latitude accuracy is 0.00074, the number of steps is set to 10). Results are shown by Tables IV and V.

TABLE IV. CENTROID TYPE REDUCTION DEFUZZIFICATION

№	Latitude / Longitude accuracy	Location
1	0.00074	84.398
2	0.00148	84.398
3	0.00222	84.398
4	0.00296	90.831
5	0.0037	99.770
6	0.00444	139.576
7	0.00518	392.995
8	0.00592	438.650
9	0.00666	460.487
10	0.0074	460.487

TABLE V. CENTER-OF-SETS TYPE REDUCTION DEFUZZIFICATION

№	Latitude / Longitude accuracy	Location
1	0.00074	84.398
2	0.00148	84.398
3	0.00222	84.398
4	0.00296	84.398
5	0.0037	84.398
6	0.00444	84.398
7	0.00518	460.487
8	0.00592	460.487
9	0.00666	460.487
10	0.0074	460.487

Here, situation retains characteristic features observed in Fig.5 and 6, i.e. centroid TRp also demonstrates better “behavior”. The line (Fig.7) grows monotonously being smooth enough, except for x-coordinates falling into the real range [0.00518, 0.00666] (approx.). Lines shown in both graphs (Fig.7,8) follow the exponential trend (the less latitude/longitude accuracy, the less location accuracy observed).

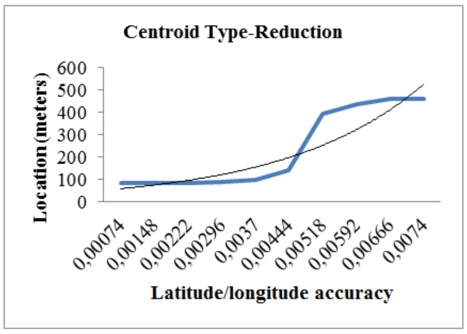


Fig. 7. Centroid type reduction method for *“latitude/ longitude accuracy”* variables

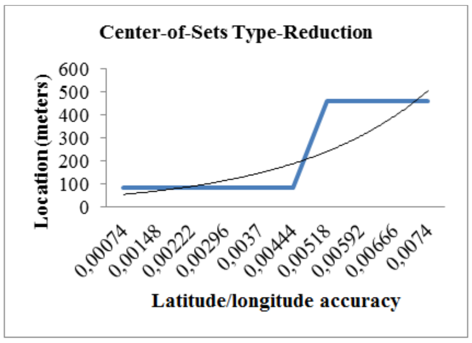


Fig. 8. Center-of-sets type reduction method for *“latitude/ longitude accuracy”* variable

We have additionally tested rules using two defuzzification methods already mentioned before, namely, (1) centroid TRp and (2) center-of-sets TRp (approaches).

C. Test 3 (checking rules)

TABLE VI. CENTROID TYPE REDUCTION DEFUZZIFICATION

№	Rule	Difference between fixed and current points (delta)	Latitude accuracy	Longitude accuracy	Location
1	1*a*	0.0032	0.00074	0.00444	139.576
2	1*b*	0.0064	0.00222	0.0037	99.770
3	1*c*	0.0096	0.00296	0.00296	139.576
4	1*d*	0.0128	0.0037	0.00222	282.0
5	1*e*	0.016	0.00444	0.00074	392.995
6	2*a*	0.0032	0.00518	0.00074	392.995
7	2*b*	0.0064	0.00444	0.00222	139.576
8-22	2*c*-5*b*	0.0096	0.00592	0.00296	392.995
23	5*c*	0.0256	0.00296	0.00296	446.153
24	5*d*	0.0288	0.0037	0.00222	441.641
25-29	5*e*-6*d*	0.032	0.00444	0.00074	392.995
30	6*e*	0.032	0.00666	0.00074	460.487
31-40	7*a*-8*e*	0.0192	0.00074	0.00666	392.995

Each rule was “fed” with 5 (five) test cases, thus each of Tables VI and VII covers $40 = 8 \times 5$ cases in total. Tests per rule are numbered in ascending order starting with [n]*a* and ending with [n]*d*, where [n] is the rule’s number (index). For example, Rule 3 corresponds to sequence of labelings 3*a*, 3*b*, 3*c*, 3*d* and 3*e* used in Tables VI and VII. Test data were generated according to intervals of each variable’s domain (the same approach as in tests 1 and 2). Step of *"delta"* changes is 0.0032, while step for the latitude and longitude variables equals to 0.00074. Input values are mixed to ensure wider coverage and variety. The last column presents location according to test values and calculation method (TRp) selected. Last column’s cells with light-grey shading determine 'At station' (≤ 282 meters) value (set), while other values show location near some station (linguistic value 'Near the station'). Both tables are wittingly shortened, because of recurrent location results.

TABLE VII. CENTER-OF-SETS TYPE REDUCTION DEFUZZIFICATION

№	Rule	Difference between fixed and current points (delta)	Latitude accuracy	Longitude accuracy	Location
1	1a	0.0032	0.00074	0.00444	84.398
2	1b	0.0064	0.00222	0.0037	84.398
3	1c	0.0096	0.00296	0.00296	84.398
4	1d	0.0128	0.0037	0.00222	275.180
5	1e	0.016	0.00444	0.00074	460.487
6	2a	0.0032	0.00518	0.00074	460.487
7	2b	0.0064	0.00444	0.00222	84.398
8-40	2c-8d	0.0096	0.00592	0.00296	460.487

A defuzzification method computes the range of possible location values according to input data provided, and the last column of tables shows a mean value of interval bounds, e.g. 139.576 is a mean of [0, 279.152] real-valued range obtained through defuzzification procedure.

V. CONCLUSION

The paper examined potentials of the modeling approach based on interval type-2 fuzzy sets (IT2FS) and conventional Mamdani fuzzy inference system (MFIS) as applied to real and topical problem related to passengers tracking in urban metro (positioning service by the example of Moscow city). Appeal and significance of developing and further analysis of such models may be of a high demand for appropriate representation of those factors that are inherently vague and uncertain. The aspects that provide for eventuality to discuss models with broad sections of stakeholders owing to model’s transparency, abilities to tune their parameters and to carry out experiments (test runs) play a sound role in theory and from practical standpoint. Empirical studies had shown that design issues concerned with linguistic variables and their labelled values (or, terms) influence significantly fuzzy model's output. Test cases presented in the paper corroborate both the applicability and relevance of fuzzy logic-based approach to various problems emerging in the field of navigational services, passenger tracking based on positional technologies. As it was mentioned in section IV, the model that makes use of IT2FS and MFIS leads at the end to resultant intervals that can be calculated in genuine mobile applications without appreciable extra costs with the object of determining the distance to notify users about their arrival (approach) to station. Hence, the developed fuzzy (prototype) model helps to estimate exemplary limits for values of each variable examined. Due to promising test results and its potential practical applicability, the model (Fig.1) will be implemented in the Android-based mobile program aimed at building routes and notifying users about their destination.

From the standpoint of further theoretical research and topic evolvement, different types of membership functions together with fine tuning of their parameters as well as alternative type reduction defuzzification methods can be considered. Besides, by way of illustration GPS technique may beat its own path in IT2FS-based models as applied to ground transportation.

REFERENCES

[1] C-Y. Chen, J.-P. Yang, G.-J. Tseng, Y.-H. Wu and R.-C. Hwang. An Indoor Positioning Technique Based on Fuzzy Logic, in *Proc. International Multi Conference of Engineers and Computer Scientists (IMECS)*, 2010, pp. 854-857.

[2] A. Teuber and B. Eissfeller. WLAN Indoor Positioning Based on Euclidean Distances and Fuzzy Logic, in *Proc. Workshop on Positioning, Navigation and Communication (WPNC)*, 2006, pp. 159-168.

[3] S. Yasunobu, S. Miyamoto and H. Ihara, A Fuzzy Control for Train Automatic Stop Control. *Transactions of the Society of Instrument and Control Engineers*, vol. E-2(1), 2002, pp. 1-9.

[4] G.J. Klir and M. Wierman, Uncertainty Formalizations, in *Uncertainty-Based Information. Elements of Generalized Information Theory*, *ser. Studies in Fuzziness and Soft Computing* (#15), 2nd ed., Physica Verlag, 1999, 168 p.

[5] L. Arigela, P. Veerendra, S. Anvesh and K. Hanuman, Mobile Phone Tracking & Positioning Techniques. *International Journal of Innovative Research in Science, Engineering and Technology,* vol.2, pp. 906-913, 2012.

[6] Gps.gov, "Official U.S. Government Information About the GPS and Related Topics, GPS Accuracy", 2017. [Online]. Available: http://www.gps.gov/systems/gps/performance/accuracy/ [Accessed 27-Feb-2017].

[7] V. Zeimpekis, P. E. Kourouthanassis and G. M. Giaglis, Mobile and Wireless Positioning Technologies, in *UNESCO Encyclopedia of Life Support Systems (EOLSS)*, vol. 6.108, EOLSS Publishers Co Ltd, 2007.

[8] Mosmetro.ru, "Metropoliten v tsifrakh", 2017. [Online]. Available: http://mosmetro.ru/press/metropoliten-v-tsifrakh/ [Accessed 28-Jan-2017] (in Russian).

[9] Nashemetro.ru, "Metro v tsifrakh.", 2017. [Online]. Available: http://nashemetro.ru/facts.shtml [Accessed 13-Jan-2017] (in Russian).

[10] J.M. Mendel, H. Hagras, W.-W. Tan, et al. *Introduction to Type-2 Fuzzy Logic Control. Theory and Applications (IEEE Press Series on Computational Intelligence),* Wiley-IEEE Press, 2014, 376 p.

[11] En.wikipedia.org, "Moscow Metro", 2017. [Online]. Available: https://en.wikipedia.org/wiki/Moscow_Metro [Accessed 25-Jan-2017].

[12] J.M. Mendel. *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*, New Jersey : Prentice Hall PTR, 2001, 576 p.

[13] N.N. Karnik, J.M. Mendel. Type-2 Fuzzy Logic Systems: Type-Reduction, in *Proc. IEEE Int. Conference on Systems, Man, and Cybernetics*, 1998, pp. 2046-2051.

[14] J.M. Mendel. Interval Type-2 Fuzzy Logic Systems and Perceptual Computers: Their Similarities and Differences, in *Advances in Type-2 Fuzzy Sets and Systems: Theory and Applications (ser. Studies in Fuzziness and Soft Computing*. A. Sadeghian, J. Mendel, H. Tahayori, Ed. Berlin : Springer Science, 2013, pp. 3-18.

A Modified Scrum Story Points Estimation Method Based on Fuzzy Logic Approach

Sofia A. Semenkovich, Olga I. Kolekonova, Konstantin Y. Degtiarev
School of Software Engineering
National Research University Higher School of Economics
3 Kochnovsky Proezd, Moscow, 125319, Russian Federation
sofya-semenkovich@yandex.ru, okolekonova@gmail.com, kdegtiarev@hse.ru

***Abstract* — Several known methods allow to estimate the overall effort(s) to be used up for the software development. The approach based on story points is preferable and quite common in the context of Scrum agile development methodology. However, it might be rather challenging for people, who are new to this methodology or to a specific Scrum team to estimate the amount of work with story points. The proposed approach involves estimation of features on the basis of linguistic terms that are both habitual and clear for everyone. The presented fuzzy inference system (Mamdani's model) makes it possible to calculate story points – the study shows empirically that beginners to Scrum methodology consider the proposed approach to be more convenient and easier in use than the 'plain' story points estimation.**

Keywords — fuzzy logic, Scrum, story points, expert estimations, aggregation of opinions, fuzzy inference system, Likert scale

I. Introduction

Many software systems relate to large-scaled and rather complex products that embrace, in particular, numerous factors to monitor and control at the development stage. Without a doubt, software development is a multifold process that essentially depends on tangled human activities, thus requiring effective management and planning [1]. Software development effort estimation acts as a key constituent of decision-making support during the process of such planning and further management. In short, effort can be defined in the context of combination «man-time» and expressed as the time (number of units) needed for a man (team's member) to complete a given task [1, 2]. Nowadays, we may address a relatively long list of recognized estimation methods aimed at evaluating efforts needed to be spent in the software development process. In fact, many efforts to categorize such methods are originating from the publications by Barry Boehm on software cost modeling and engineering economics in the early eighties of the previous century. We cannot talk about «the best» from all conceivable standpoints classification, but in rough outline such methods can be divided into three aggregative categories, namely: methods based on expert subjective estimates and views (non-model based methods), formal estimation methods that are grounded on specific or generic models, and combined (or, composite) methods built upon joint use of analysis and processing of available from different sources data along with expert estimates [3]. Amongst others, the first category takes in such approaches as planning poker (also known as Scrum poker) and Wideband Delphi, two similar methods where the provided estimations are based on judgments and expressed opinions of project's stakeholders [1]. In formal estimation models (e.g. Constructive Cost Model (COCOMO), COCOMO II as a generalization of COCOMO, weighted micro function points (WMFP), SLIM, use case modeling, story points) formulas and/or results derived from earlier implemented projects are used.

In the present paper, we consider the method of estimation with story points in the context of Scrum, an agile flexible framework to manage the process of software development. The main goal of Scrum is to deliver new software capability (features) every 1-2 weeks (the duration can be extended), each new version includes the most important features for Product Owner, thus allowing to inspect and adapt product to current conditions. The main Scrum characteristic of the estimation process is that Product Owner defines priorities for the features because the product should be maintained in a tested/integrated state every Sprint (i.e. fixed number of days team works together to produce beforehand coordinated changes in the product), so the work should be broken down to pieces/stages [4]. In case of proper compliance with other Agile principles, the release deadline cannot be missed by the team; if the features were evaluated incorrectly by some reasons, skipping over the less important tasks can be the only noticeable disadvantage as compared to waterfall or pseudo-Agile teams' experience.

In contrast with other approaches, Scrum is concerned with two main factors that are important in estimating development efforts. Firstly, the responsibility for the product falls on the shoulders of the whole team rather than individuals. It means that there are no gradations like «*my work*» and «*your work*». The framework attracts attention to cumulative effort(s) per Product Backlog' Item rather than individual effort(s) per feature. Secondly, the tasks are estimated in a relative manner, i.e. they are assessed (compared to each other) in terms of relative units, but not absolute ones. Thus, story points may be employed as such units of measure to express an estimate of the overall effort required to fully implement a product backlog item or any other piece of work [5]. As it is noticed by Joshua Kerievsky [6], "… Many say that story points make us better at estimating because we're estimating the size of work, rather than the time it takes to complete it; … in 2005, one of our customers found story points to be so confusing that he renamed them NUTs (Nebulous Units of Time)".

Such witty testimonial inherently expresses the attitude of newcomers to Scrum development methodology towards story points, their 'fear' of commonly used phrases and statements: «number of points per sprint», or «the estimate in story points is better than estimate in hours», etc. There are many helpful and well-composed electronic and printed sources dedicated to Scrum's set of principles and practices aimed at developing complex systems – books and articles by J. Sutherland, C. Sims, A. Stellman, guides, reports, tutorials on Scrum and other Agile methodologies by AgileRussia.ru, Scrum Alliance®, training courses from ScrumTrek, Scrum.org, LuxSoft, to name a few. Even cursory glance at results of Google search gives cause for being not fully confident indeed in the conception of various word-combinations related to story points enquiries, e.g. «they are cheaper than hours», «relative unit of measure», «estimate of effort», and the like.

In brief, story points are founded on "a short description of a set of features called user stories"; each such story will have a set of story points [7]. When we estimate features with story points, we assign a point value to each item. The raw values we assign are unimportant (we can talk about unit of measure that team's members agreed on), what matters are the *relative values*. A story that is assigned a value of 2 should require twice as much effort as a story that is assigned a value of 1, and it also constitutes two thirds of a story that is estimated at the level of 3 story points. Because story points represent the effort(s) to develop a story, a team's estimate must cover every aspect that can affect the effort. In general, they bring together as a single whole the amount of work to do, the complexity of the work, any risk or uncertainty in doing the work.

Our research proposes to simplify the process of estimating features with the help of story points. For most of people it is rather confusing or even difficult to combine three aforesaid components into one in their mind and give an approximate resultant value. Instead of evaluating the features with story points, we assume that each member of the Scrum team (e.g. expert) provides his/her opinion regarding two factors, namely, these are the amount of work to do and its complexity. Besides, the experts should also specify the level (or, degrees) of their confidence in both such estimates. Experts operate with preset collection of linguistic terms expressed as words or phrases of the natural language. These verbal units are converted after that to proper fuzzy sets used in further processing. The latter provides application of fuzzy inference system (FIS) for each expert's estimations and aggregation of the results obtained into one outcome. What are the reasons to resort to the help of fuzzy approach? Well, we can partly refer to [8] saying that "many fuzzy categories described linguistically appear to be more informative than precise descriptions".

On top of that, a short survey was also conducted with the aim to figure out the opinions of four different groups of people on proposed approach. The core of this activity is the comparison of story points obtained in "experimental" manner and regular story points estimation.

The rest of the paper is organized as follows: section II presents basic definitions, terms (type-1 fuzzy set, linguistic variable, inference system, defuzzification, aggregation of estimates) that are used in the subsequent parts of the paper. The proposed approach to obtain story point-based estimates on the basis of defined input variables of Mamdani's fuzzy inference system is discussed and visualized in section III. The results of conducted experiment (empirical study) with several groups of people having different practical skills relative to use of story points estimations are discussed in the section IV. Concluding remarks are drawn in section V.

II. Basic definitions and general comments

In clear majority of cases humans express their opinions and judgments using statements of natural language; many things that are thus heard or said are vague to a variable degree. According to Stanford Encyclopedia of Philosophy, a term «is vague to the extent that it has borderline cases», and the latter acquires special significance in relation to the vagueness that has to be modeled in adequate way for the case under consideration. In general, such task appears simple enough only at the first glance, and one of practical approaches, at least, from perception-based point of view, relates to fuzzy logic (FL) methodology. It provides ample means to model the perceived meaning of words/phrases conveying the experts opinions (estimates) in a graded fashion. Following seminal paper "Fuzzy Sets" by L.Zadeh [9], the concept of fuzzy set constitutes a class of objects with continuum membership grades.

Definition 1. Let U be a set of elements (objects) that are denoted generically as x (U={x}); fuzzy set $A \subseteq U$ is a set of ordered pairs $\{(x, \mu_A(x))\}$, where mapping $\mu_A : x \to [0,1]$ is a (type-1) membership function of a fuzzy set A. Value $\mu_A(x)$ is a degree (grade) of membership of x in the set A.

In many situations the shape of membership function can be set by a specialist (expert, domain engineer); such manual tuning of function's parameters turns out to be sufficient at the initial stages of model's development and processing. Thus, piecewise linear functions are often chosen due to their usability, expressive power in grasping thoroughly both the knowledge and human's perception of situation, as well as computational efficiency.

Definition 2. Trapezoidal membership function [10] is defined by a 4-tuple (a_1, a_2, a_3, a_4) of its parameters in the following way:

$$\mu_A(x) = \begin{cases} 0, \ x \in (-\infty, a_1) \\ (x - a_1)/(a_2 - a_1), \ x \in [a_1, a_2] \\ 1, \ x \in [a_1, a_2] \\ (a_4 - x)/(a_4 - a_3), \ x \in [a_3, a_4] \\ 0, \ x \in (a_4, +\infty) \end{cases} \qquad (1)$$

Normalized trapezoidal (and triangular with values $a_2 = a_3$)

functions having height $h = \max(\mu_A(x)) = 1,\ \forall x \in U \subset \mathbb{R}^1$, often describe values in the form «close to *b*», «around *b*», where *b* is either a crisp real number $b_{\text{val}} \in \mathbb{R}^1$, or the interval $\left[b_{\text{val}}^{(1)}, b_{\text{val}}^{(2)}\right] \subset \mathbb{R}^1$.

Definition 3. A linguistic variable is characterized by a 5-tuple $\langle L_v, T(L_v), U, R_{\text{syn}}, R_{\text{sem}} \rangle$, where L_v is the name of the variable (e.g. $L_v \equiv \text{"complexity of work"}$), $T(L_v)$ is the set formed by labels of variable's L_v linguistic values $l_1,..,l_n$ (term-set of L_v; e.g. 'easy', 'normal', 'difficult', etc.). These names are generated using syntactic rule R_{syn}, whereas the meaning $R_{\text{sem}}(l_i)$ is associated with each value $l_i, i = \overline{1,n}$, from $T(L_v)$ by means of semantic rule R_{sem}; $R_{\text{sem}}(l_i)$ is a fuzzy set (respective membership function) defined on a universe of discourse U. Linguistic modifiers (or, so-called hedges) 'very', 'more or less' and the like, together with logical connectives 'and', 'or' and negation 'not' are treated as special type operators that modify the primordial meaning of primary values (terms) $l_1,..,l_n$. It results in altered shape of membership functions representing $l_1^{\text{mod}},..,l_n^{\text{mod}}$ [11].

Definition 4. The fuzzy inference is a process of deriving conclusion from given premises and system's inputs (or, given fact), for which compositional rule of inference (CRI) serves as a core. CRI can be viewed as a generalization of modus ponens argument scheme (the mode that affirms). The premises are represented as a set of If-Then rules forming knowledge base Ω, e.g. If x is A_i Then y is B_i, $i = \overline{1,m}$, as a basic case ($A_i \to B_i$, i.e. A_i implies B_i) – potentially, such rules may have more complex appearance. Mamdani-type fuzzy inference system (FIS) proposed and evolved by E.H. Mamdani and S. Assilian in 1975 owing to the examination of fuzzy logic controller can be expressed as $B'(y) = \bigcup_{x \in U_1} A'(x) \wedge \tilde{R}(x,y)$, where relation $\tilde{R}(x,y)$ is calculated as follows: $\tilde{R}(x,y) = \bigcup_{i=1}^{m} A_i(x) \wedge B_i(y)$, where A_i and B_i are (type-1) fuzzy sets, $A_i \subset U_1$, $B_i \subset U_2$.

The knowledge base Ω represented as a set of If-Then rules constitutes rather convenient and transparent form to express individual expert conceptions of phenomenon under study as well as perceptions of a group of specialists. On the whole, model Ω is a handy tool to discuss hypotheses (under potential tuning up rules and initially set parameters of fuzzy sets, if needed) and to make final decisions.

The process of representing initial data (e.g. linguistic values) as membership functions is called fuzzification; most of applications require to perform at final stages the opposite translation from fuzzy functional forms to crisp values; the latter act as representatives of corresponding fuzzy sets. This is achieved through defuzzification procedures, and one of commonly utilized method is called Center Of Area (COA). It stipulates calculation of the resultant value res^* by way of

$$\text{res}^* = \frac{\int_U x \cdot \mu(x)dx}{\int_U \mu(x)dx} \tag{2}$$

The intersection (*AND*) and union (*OR*) operations that are used in computational schemes with fuzzy sets are expressed as functions called t-norms $T(\cdot)$ and s-norms $S(\cdot)$, accordingly [12]. Different types of $T(\cdot)$ and $S(\cdot)$ are presented and discussed at length in the literature (e.g. [13]) – without loss of generality, in the paper we use standard *min* and *max* operators:

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mathit{min}(\mu_A(x), \mu_B(x)) \tag{3}$$

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mathit{max}(\mu_A(x), \mu_B(x)) \tag{4}$$

It is worth noting that story points are crisp numbers, because they appear to be the most convenient and easy "units" to compare and interpret by Scrum team members as compared to, for instance, numeric intervals. Thus, crisp numbers are associated with story points, which help to rank features in compliance with efforts required to implement them. As it was mentioned before, the valuable source of information are expert judgments (estimations), and once all such estimations are obtained, they should be aggregated to form conjoint opinion. Such activity can be performed by a dedicated person called *analyst*. With this aim in mind, two methods of aggregation are used in the paper.

The first method of aggregation is applied when all estimations elicited from Scrum team members (experts) are different, with one minimum and one maximum denoting left and rights extremities in the resultant sequence. For example, if it is of a form 10, 25, 46, 34, 30, 47, 28, simple expression allows to calculate the aggregated estimate:

$$e_{\text{agr}} = \left(\sum_{i \neq i_{\min}, i_{\max}} e_i - e_{\min} - e_{\max}\right) \Big/ \left(n_e - 2\right) \tag{5}$$

where e_{agr} is the aggregated estimate, $e_{\min}$ and $e_{\max}$ are minimum and maximum values among obtained estimations, respectively, n_e is the total number of values in the sequence, summation goes over all estimations excluding $e_{\min}$ and $e_{\max}$.

The second aggregation method (weighted arithmetic mean) can be used in situation of appearance of recurring experts' estimations as in the case of values 10, 25, 10, 34, 25, 47, 28; such outcomes (with repetitions) are rather practicable, so they should be addressed reasonably enough. If R_e is the most recurring estimate (conditional mean) observed in the numeric sequence, then e_{agr} can be obtained as follows:

$$e_{\text{agr}} = R_e - \left(\sum_i \left((e_i - R_e) \cdot f_i\right)\right) \Big/ n_e \tag{6}$$

where f_i is the frequency of e_i occurrence in the row of estimations provided.

All prepared comments allow to proceed to approach that may assist people who are new to Scrum methodology (or, they are newcomers to a specific Scrum team) and who do not fully understand how they can estimate the amount of work to do on the base of story points. The central idea of such approach relates to a natural course, i.e. story points seem brittle and a bit confusing – fine, try in that case to estimate how much certain part of work will take making good use of terms you are familiar with. The aforesaid definitions simplify the perception of the following material, and they should not be considered as an extra "difficulty" to tackle on top of Scrum methodology itself; «*such overload is a bit too thick!*» – the reader may exclaim. We think, in no way, as long as all necessary (not very complex) calculations can be done by analysts; in other respects, interviewing and grasping the verbal statements expressing the results (what is said) in pretty understandable form are natural and plain day-to-day human activities.

III. Expert Opinions and Levels of Confidence – Modified Likert Scale and Fuzzy Approach

Suppose that through talks and consultations with experts, the analyst collected the opinions (estimations) of several experts on certain feature expressing how much work, reasoning from their understanding and perception, they'll have to do to implement this feature, complexity of the work and their level of confidence about each of these estimations. After fuzzification of verbal data obtained and applying fuzzy rules, the aggregated result is converted to story points; the latter can be used at subsequent stages in any project management system.

As it was already mentioned earlier, the expert puts his/her opinion concerning complexity, amount of work as well as degree of confidence in estimation expressed in linguistic forms (statements) [14]. For example, the expert may say the following: «*I'm quite sure that this feature will be difficult to implement, besides I must do a large amount of work to implement this feature, however, I'm not very sure about it*». From this sentence, we can pick out the following pairs of linguistic terms, namely: 'difficult' → 'quite sure' and 'large' → 'not very sure'. With such estimations in mind (and their formal representation by way of fuzzy sets), we'll be able to proceed to the construction of corresponding fuzzy rules [15].

TABLE I. Parameters of Trapezoidal Membership Functions Representing Values of Term-Sets

The amount of work (set T(A))	The complexity of work (set T(C))	The overall effort (set T(E))
value 'very small' (1,1,5,20)	value 'very easy' (1,1,5,20)	value 'tiny' (1,1,5,20)
value 'small' (5,15,30,40)	value 'easy' (5,15,30,40)	value 'little' (5,15,30,40)
value 'medium' (25,40,60,75)	value 'normal' (25,40,60,75)	value 'average' (25,40,60,75)
value 'large' (60,70,85,95)	value 'difficult' (60,70,85,95)	value 'big' (60,70,85,95)
value 'very large' (80,95,100,100)	value 'very difficult' (80,95,100,100)	value 'huge' (80,95,100,100)

It is commonly advised to use the interval [1,100] to represent story points estimations, so we direct our attention to the same extreme points 1 and 100 to define the universe U to specify fuzzy sets [4]. The amount of work to do, the complexity of the work and the degrees of confidence are considered as system's input variables, whereas the overall (combined) effort is taken as an output variable. Thus, the following linguistic variables $L_v^{(i)}$ denoted as A, C and E and their values (labels of linguistic terms) are considered [11]:

$L_v^{(1)} = A \equiv$ "amount of work to do",

$L_v^{(2)} = C \equiv$ "complexity of work" (Fig. 1),

$L_v^{(3)} = E \equiv$ "the overall (combined) effort" (Fig. 2), where

T(A) = { 'very small', 'small', 'medium', 'large', 'very large' },
T(C) = { 'very easy', 'easy', 'normal', 'difficult', 'very difficult' },
T(E) = { 'tiny', 'little', 'average', 'big', 'huge' }.

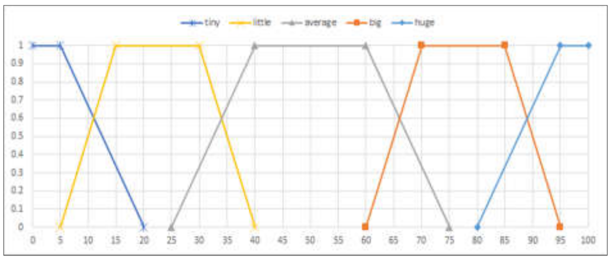


Fig. 1. Linguistic variable C = "complexity of work"

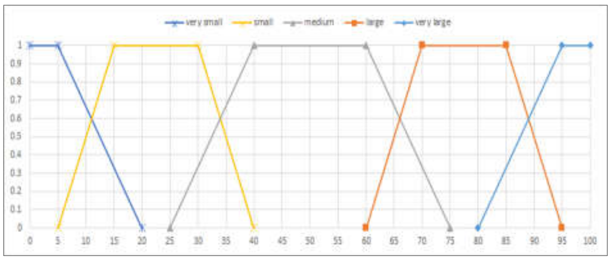


Fig. 2. Linguistic variable E = "the overall (combined) effort"

TABLE II. Correspondence Between Levels of Confidence and Their Values

Level of confidence (linguistic term)	Value
'not sure at all'	0.05
'almost not sure'	0.15
'not very sure'	0.35
'more or less sure'	0.5
'sure'	0.65
'quite sure'	0.8
'definitely sure'	0.95
'extremely sure'	1

After consultations with experts, the analyst (and his group) defines the parameters of trapezoidal membership functions (1) to represent formally values of term-sets T(A), T(C) and T(E) fuzzy sets as shown in Table I. For example, if expert says something like «*... this feature is hard to implement, but I must do small amount of work*», we select primary linguistic values 'small' from the set T(A) and 'difficult' – from T(C).

The parameters of membership functions (Table I) were chosen empirically, although slight alterations of values within certain bounds ($\pm\varepsilon_i$, $i=\overline{1,k}$, k is the number of deliberate assortments of such deviations on all terms of sets T(·)) turn out to be allowable. Such "mobility" of value ranges may bring to the advisability to consider further on type-2 interval fuzzy sets – unlike type-1 sets, they enable to express the uncertainty about the membership grades of elements on the domain considered.

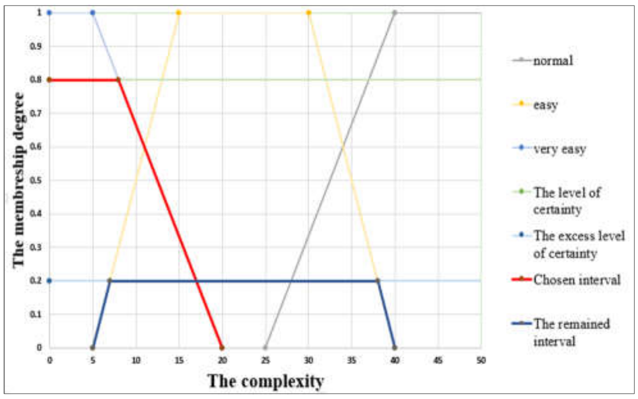


Fig. 3. The distribution of confidence levels (fuzzification stage)

TABLE III. THE ACCORDANCE OF THE AMOUNT OF WORK AND THE COMPLEXITY OF WORK TO OVERALL EFFORT

A\C	**very easy**	**easy**	**normal**	**difficult**	**very difficult**
very small	tiny	tiny	little	average	average
small	tiny	little	little	average	average
medium	little	little	average	big	big
large	average	average	big	big	huge
very large	average	average	big	huge	huge

The next step is to relate the level of confidence to fuzzy set being thought about. The ideas and views concerning Likert scale (psychometric response scale suggested by American sociologist Rensis Likert in 1932) allow to come out with relatively simple scheme to use in aforesaid task. Following QingLi, the level of agreement (LA) as an estimate within the range [0,1] can be associated with the membership degree (as an option, terms 'strongly agree', 'agree', 'neither agree, nor disagree', 'disagree' and 'strongly disagree' can be in use) [16]. The sum of LA for all options is equal to 1. In the case considered, the option provided by an expert and the level of agreement is experts' levels of confidence are shown in Table II. However, if expert's level of confidence is not 'extremely sure', we are facing with the excess of LA. Thus, it can be suggested to distribute emergent excess between the nearest neighbors of the option selected by the expert. If there are two nearest neighbors, they both will get half of the excess observed; if there is only one nearest neighbor, it will get the whole amount of excess. In the paper, we use crisp numbers to represent level of confidence' values as the starting point of our approach. These values are based on the results of survey – opinions of approximately 50 people concerning the correspondence between linguistic values (labels) of confidence level and their actual mapped numbers were first

For example, if expert says that his/her level of confidence can be expressed as 'quite sure' (i.e. expert explains that «*…I'm quite sure that…*»), and the feature under consideration is very easy to implement, we choose fuzzy number representing term 'very easy' and define degree of membership as being equal to 0.8 – it is the value of choice. Thus, the excess level of confidence comes to 0.2, and it is handed over to the nearest neighbor of the term 'very easy', which is 'easy'. This distribution of confidence levels is shown graphically in Figure 3.

Based on the information and knowledge elicited from experts, we may design a set of fuzzy rules (fuzzy rule-base). The amount of work to be done and the complexity of work act as input variables, and their combination result in the value of the overall effort. In general, these rules reflect the perceptions of experts, their feelings and conclusions drawn regarding situation given. For instance, a "typical" question may look as follows: «*How much will it take in the sense of overall effort to accomplish a 'very easy' task that needs just 'medium' amount of work to be done*». The short version of the rule-base is represented in Table III, whereas the full set is provided below (rules Ri, $i=\overline{1,5}$). From the very outset, there were 25 rules (one rule for each combination of the amount of work (*A*) and the complexity of work (*C*)). Later, they were combined on the base of resulting value of overall effort, and only five rules R1,…,R5 were retained.

- rule R1:
IF amount is 'very small' ***AND*** complexity is 'very easy' ***OR***
amount is 'very small' ***AND*** complexity is 'easy' ***OR***
amount is 'small' ***AND*** complexity is 'very easy',
THEN effort is 'tiny'

- rule R2:
IF amount is 'very small' ***AND*** complexity is 'normal' ***OR***
amount is 'small' ***AND*** complexity is 'easy' ***OR***
amount is 'small' ***AND*** complexity is 'normal' ***OR***
amount is 'medium' ***AND*** complexity is 'very easy' ***OR***
amount is 'medium' ***AND*** complexity is 'easy',
THEN effort is 'little'

- rule R3:
IF amount is 'very small' ***AND*** complexity is 'difficult' ***OR***
amount is 'very small' ***AND*** complexity is 'very difficult' ***OR***
amount is 'small' ***AND*** complexity is 'difficult' ***OR***
amount is 'small' ***AND*** complexity is 'very difficult' ***OR***
amount is 'medium' ***AND*** complexity is 'normal' ***OR***
amount is 'large' ***AND*** complexity is 'very easy' ***OR***
amount is 'large' ***AND*** complexity is 'easy' ***OR***
amount is 'very large' ***AND*** complexity is 'very easy' ***OR***
amount is 'very large' ***AND*** complexity is 'easy',
THEN effort is 'average'

- rule R4:
IF amount is 'medium' ***AND*** complexity is 'difficult' ***OR***
amount is 'medium' ***AND*** complexity is 'very difficult' ***OR***

amount is 'large' ***AND*** complexity is 'normal' ***OR***
amount is 'large' ***AND*** complexity is 'difficult' ***OR***
amount is 'very large' ***AND*** complexity is 'normal',
<u>THEN</u> effort is 'big'

- rule R5:
<u>IF</u> amount is 'large' ***AND*** complexity is 'very difficult' ***OR***
amount is 'very large' ***AND*** complexity is 'difficult' ***OR***
amount is 'very large' ***AND*** complexity is 'very difficult',
<u>THEN</u> effort is 'huge'.

Let's consider the following expert's verdict: «*Well, I am quite sure that this {feature} is easy to implement; to tell the truth, I'm also more or less sure that it requires a large amount of work to do*». From this statement, we can extract the following pairs of linguistic terms: 'easy' → 'quite sure' and 'large' → 'more or less sure'. Membership degrees in use are summarized in Tables IV and V (elements of T(A) and T(C) – five terms in each case):

TABLE IV. MEMBERSHIP DEGREES OF THE COMPLEXITY (EXAMPLE)

The complexity of work (set T(C)**)**	**Membership degree**
value 'very easy'	0.1
value 'easy'	0.8
value 'normal'	0.1
value 'difficult'	0
value 'very difficult'	0

TABLE V. MEMBERSHIP DEGREES OF THE AMOUNT OF WORK (EXAMPLE)

The amount of work (set T(A)**)**	**Membership degree**
value 'very small'	0
value 'small'	0
value 'medium'	0.25
value 'large'	0.5
value 'very large'	0.25

In this case, fuzzy rules R2, R3 and R4 will give non-zero resultant value. As already stated above, Mamdani inference system (FIS) is used in the experiments – it allows to obtain an output in the form of fuzzy set. Rules R2, R3 and R4 "fire", thus ensuring non-zero results; in compliance with (3) and (4), we arrive at the following:
R2: $max\left(min(0.1, 0.25), min(0.8, 0.25)\right) = 0.25$ – membership degree that corresponds to the term 'little' (element of T(E)),
R3: $max = 0.5$ – membership degree that corresponds to the term 'average' (element of T(E)),
R4: $max\left(min(0.1, 0.5), min(0.1, 0.25)\right) = 0.1$ (label of the term 'big' as the element of T(E)).
COA (Center Of Area) method (2) is applied to obtain crisp result. According to equation (2), the output value equals to approx. 45 story points as shown in Fig. 4.
In Scrum story points estimation' approach the experts often aggregate their opinions using the method of planning poker. It relies on collective judgments (several rounds may become necessary until experts make an agreement) and tries to avoid "pointless haggling over small differences" by compelling to use estimation value from a set of sharply defined distinct values [17]. All participants (they can also be called estimators) secretly write down their estimations in story points on preprepared cards, and then all cards are laid on the table at one time. If all participants select the same value, this value becomes the feature estimation. If not, each expert one after another explains his/her reasons in showing preference for specific value provided, especially when the choice is fixed upon the highest and the lowest estimators in the set. Afterwards, the process is reiterated, i.e. experts vote again, planning poker goes on. It continues until estimators arrive at the agreement.

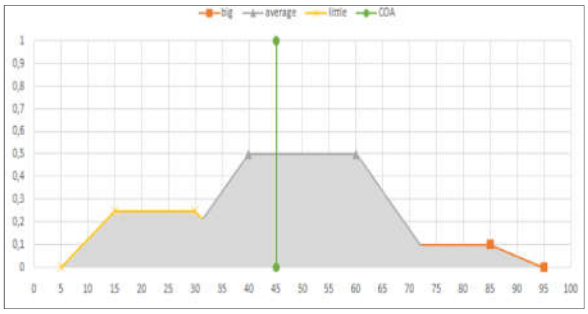


Fig. 4. The result of defuzzification (COA, approx. 45 points)

As regards the aggregation procedure, two approaches mentioned earlier are used. The exact way to calculate the aggregated opinion based on estimations expressed is chosen according to simple rule: if some of them (estimations) are repeated, the equation (6) is used; otherwise, the equation (5) is preferred.

IV. RESULTS OF EXPERIMENT – DIFFERENT GROUPS OF POTENTIAL USERS. DOES THE PROPOSED METHOD WORK?

For the sake of completeness, we have asked several groups of people about their views regarding proposed method (its details were discussed with persons concerned in advance). **Group 1** consisted of those people who have worked with story points for a long time. Those delegates who worked with story points before for relatively short-term period formed **group 2**, while those who know what story points are, but have never used them earlier found themselves in the **3rd group**. Finally, people who never even heard of story points fell into **group 4**. As a result, opinions stated below were emphasized (single form of statements are cited for convenience):

(1) **group 1**: «*... I personally consider story points to be the most effective and quite fast way of evaluating features. I make almost no mistakes in estimating features now, and I can adapt myself in new projects in a short time. Your approach is not useful for me now, though I think it might be helpful at the beginning of (my) career*»,
(2) **group 2**: «*... As for me, it took about two months to fully understood the concept of story points, but even now I sometimes make mistakes while estimating*

features in terms of story points. Today I believe that estimating in story points is more convenient than estimating in hours or some other units. I'm quite experienced member of the currently ongoing project, and I don't need your approach now, though I could still use it, if I have to get the feel of some new project later on»,

(3) **group 3**: «*... I have heard that story points exist, and that they are used in project estimation, though I have no experience of participating in real projects, where story points were adopted. I think that your approach is better for me right now than story points in their "pure" appearance as I understand it more clearly as compared to story points per se*»,

(4) **group 4**: «*... Oh, I have no idea what are these "story points" are, so obviously, I better prefer to give my opinion on how much work I will have to do to implement the feature, or how difficult this work seems to me*».

Afterwards, we gave people a description of the project (Android App "VR Quest in city" and its features planned for implementation) was introduced to people who took part in the interview session. They were asked to estimate these features both (A) in terms of "plain" story points and (B) using proposed approach.

TABLE VI. THE RESULTS OF THE CONDUCTED EXPERIMENT

Feature name		**gr. 1**	**gr. 2**	**gr. 3**	**gr. 4**
Create a login form	story points	30	28	40	45
	our approach	35	37	30	28
Find a quest with specific parameters	story points	27	30	55	60
	our approach	29	24	27	30
Save/load a quest	story points	20	25	35	45
	our approach	18	22	20	19
Begin a quest walkthrough	story points	15	18	25	40
	our approach	16	14	15	17
Buy quests in local currency	story points	50	48	75	85
	our approach	52	55	50	45

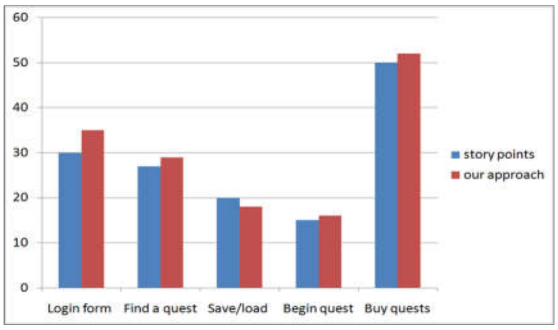


Fig. 5. The results of the conducted experiment as applied to group 1

As shown in Table VI and Fig. 5-6 (data obtained for groups 1 and 4 only are visualized), the results of basic story points estimation for the group 1 (participants in this group know how to estimate features in story points), differ not appreciably from the results revealed by proposed approach. It can be treated as initial piece of empirical evidence of the fact that our method is relevant enough and can be used for feature estimation and further elaboration. Moreover, results in both groups 3 and 4 (members of these groups have never used story points before) are substantially different in case of our method as compared with basic story points estimation' approach. This can be attributed to the marked fact that people do not really understand what story points are in the context of non-using them earlier. This is an extra argument in favor of potential utility of the proposed method for those people who are new to Scrum.

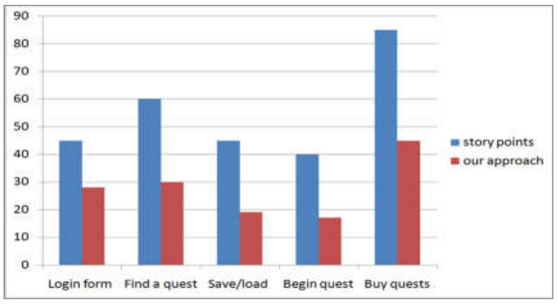


Fig. 6. The results of the conducted experiment as applied to group 4

Taking story points estimates as landmarks, the Root Mean Square Error is growing steadily from 2.76 for group 1 to 28.26 for group 4 (for groups 3 and 4 the error values are equal to 6.18 and 19.15, correspondingly). The MAD measure, i.e. the size of deviation in units of landmarks from values calculated with the help of proposed approach ((A) and (B) estimates, Table VI), is progressing from 2.4 (group 1) to 27.2 (group 4), while values of 5.8 and 17.6 stand for groups 3 and 4, accordingly. These error values show certain tendency of drawing groups 1 and 2 together along with more perceptible isolation (or, distancing) of «groups 3 and 4» bundle from the practical standpoint of both perception and acceptance of story point-based estimation approach. However, even against a background of such observation, group 3 reveals some positive "detachment" toward group 4. In aggregate, we may conclude that the proposed method has rather tangible effect (in decreasing sequence) on group 1, group 2 and group 3 just "touching" the latter in passing. To the opinion of authors, it can be treated as encouraging sign that is incentive to continue research in this direction.

V. CONCLUSION

A novel approach that relates to feature estimation in terms of story points was presented in the paper. The natural idea behind the approach reflects the fact that people may estimate their perception (ideas) concerning the *complexity* of implementation of certain product's feature to be and the *amount of work* to be done to develop this feature. Besides, they can also specify the level of their confidence (or, confidence degree) in evaluation provided. Fuzzy inference scheme lays both solid and transparent groundwork for converting aforesaid input information (data) to the number of story points that can be utilized in the software project management (SPM) at a later stage.

To the opinion of authors, this approach allows people to adapt to Scrum more smoothly, with better understanding of what is implied by story points, grasping the general idea and learning faster their use in practice. The experimental study of the proposed method has shown results approaching the estimations provided by Scrum experts who have been working in real projects and making use of story points for several years. According to survey conducted, such approach can be successfully applied by Scrum newbies, since it is more convenient for people who just make up with story points estimations.

It must be noted that full awareness of strong and weak points of the proposed approach reasoning from one example (project) cannot be realized entirely. Therefore, a sequel of empirical studies and active cooperation with Scrum teams may result in enhancement of the approach. One thing is just to mention that the method seems both promising and handy, but it's quite another matter to make it applicable in practice because of convenience and clearness, at least, as a part of induction stage of the "immersion" to Scrum. Transparent ideas of fuzzy logic are very much to the point here.

Further steps can be associated with intensive studies of more complicated methods of aggregation of the experts' opinions – in particular, they may consider the level (or, weight) of professional qualification of domain experts drawn into project activity. Currently a program's prototype to support (implement) the approach discussed in the paper is under development. The present-day agenda also covers the development of plugin for JIRA tracking system. It is also worth mentioning that certain refinements and changes of the proposed approach can be done at the theoretical level either – some of them are visible enough at present. For instance, the confidence degree values can be represented as intervals, i.e. a form of uncertainty/vagueness expression at the lowest level of comprehension. Such intervals may come about as an effect of possible discord concerning the choice of crisp values shown in the Table II. For the time being, these values may be treated as rough aggregated estimates underlying the computational steps of the discussed approach. Besides, the transition from intervals to type-1 fuzzy sets is also an explicable option to consider. Fuzzy set can be decomposed into a series of nested crisp intervals (so-called α-cuts of a fuzzy set), and this fact can be effectively used in algorithms. Without confining ourselves to just modeling linguistic terms that stand for confidence levels in use, type-2 fuzzy sets and systems are also regarded as "right" candidates for expansion research efforts in a given problem.

References

[1] Trendowicz A., 2013. Software Cost Estimation, Benchmarking, and Risk Assessment: The Software Decision-Makers' Guide to Predictable Software Development, Springer-Verlag
[2] Živadinović J., Medić Z., Maksimović D., et al., 2011. Methods of Effort Estimation in Software Engineering // Proc. Int. Symposium Engineering Management and Competitiveness (EMC), 417–422. [9] Zadeh L.A., 1965. Fuzzy Sets, Information and Control, #8, 338–353.
[3] Briand L.C., Wieczorek I. Resource Estimation in Software Engineering // Int. Software Engineering Research Network, TR ISERN 00-05, web-resource: https://pdfs.semanticscholar.org/943d/a2bb363c06319218ee204622bb10f816490f.pdf (access date 24.02.2017)
[4] Shivangi S., Umesh K., 2016. Review of Various Software Cost Estimation Techniques // International Journal of Computer Applications, vol. 141, 31–34.
[5] Colomo-Palacios R. González-Carrasco I., et al., 2012. Resyster: A Hybrid Recommender System for Scrum Team Roles based on Fuzzy and Rough Sets // Int. Journal Appl. Math. Comput. Science, 2012, Vol. 22, No. 4, 801–816.
[6] Industrial Logic site: Stop Using Story Points, Kerievsky J. (blog), 2012, web-resource: https://www.industriallogic.com/blog/stop-using-story-points/ (access date 24.02.2017)
[7] Pries K.H., Quigley J., 2010. Scrum Project Management, CRC Press
[8] Aliev R.A., Aliyev R.R., 2001. Soft Computing and Its Applications, World Scientific
[10] Bingyi K., Daijun W., Li Y., Deng Y., 2012. A Method of Converting Z-Number to Classical Fuzzy Number // Journal of Information & Computational Science, 9, #3, 703–709.
[11] Zadeh L.A., 1975. The Concept of a Linguistic Variable and Its Application to Approximate Reasoning - I // Information Sciences, vol. 8, no. 3, 199–249.
[12] Fuzzy Logic Fundamentals, Pearson Education, 2001, Ch.3, 61–99, web-resource: http://ptgmedia.pearsoncmg.com/images/0135705991/samplechapter/0135705991.pdf (access date 21.03.2017)
[13] Klir G.J., Bo Yuan., 1995. Fuzzy Sets and Fuzzy Logic: Theory and Applications, 1st ed., Prentice Hall
[14] Zadeh L.A., 1996. Fuzzy logic = Computing with Words // IEEE Trans. Fuzzy Systems, vol. 4, no. 2, 103–111.
[15] Zadeh L.A., 1992. Fuzzy Logic and the Calculus of Fuzzy If-Then Rules // Proc. 22nd Intl. Symp. on Multiple-Valued Logic, Los Alamitos, CA: IEEE Computer Society Press, 480–480.
[16] Quing L., 2013. A Novel Likert Scale Based on Fuzzy Sets Theory // Expert Systems with Applications, vol. 40, #5, 1609–1618.
[17] Meyer B., 2014. Agile! The Good, the Hype and the Ugly, Springer Int.

The Mixed Chinese Postman Problem

Maria Gordenko
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
mkgordenko@edu.hse.ru

Scientific Advisor: Prof. Sergey Avdoshin
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
savdoshin@hse.ru

Abstract— **The Mixed Chinese Postman Problem (MCPP) is to find a minimum shortest tour traversed each directed and undirected edges at least once. The problem is NP-hard. However, the mixed case of the problem has many potentially useful applications. This paper review the related works as well as mathematical formulation of the problem. The algorithms for the MCPP are based on reduction to classical Arc Routing Problems (ARP) are provided. Experimental results of algorithms' efficiency are also pointed out.**

Keywords—Mixed Chinise Postman Problem, Arc Routing Problem, heuristic algorithm, Traveling Salesman Problem

I. Introduction

The Chinese Postman Problem (CPP) was originally studied by the Chinese mathematician Kwan Mei-Ko in 1962 on the example of the rural postman problem [1]. A problem is called the CPP after Kwan Mei-Ko [2].

In the modern world, the number of companies and industries that are interested in building an optimal route of product delivery is growing. For example, the postman delivering letters or leaflets wants to know the optimal route that traverses every street in the given area, starting and ending at the office [3].

Apart from the traditional application of the CPP to solving the routing problems such as path planning of snowplows or serving teams, there is a wide range of applications including robot exploration, testing web site usability and finding broken links [3].

There are various classifications of the CPP. This problem can be applied for a directed, undirected, mixed graph, or in a multigraph (a graph with parallel directed and undirected edges). The CPP can also be closed (the postman should return to the starting point) or open (starting and ending points can be different). The problem in directed or undirected graph has exact algorithms and may be solved in polynomial time. The mixed case is NP-hard and there are no polynomial-time algorithms for solving the CPP in mixed graph or multigraph exactly [4, 3].

In this paper, heuristic algorithms for the mixed case are described and assessed. The mixed CPP (MCPP) is a simply-stated problem, which has many useful applications, but has no exact algorithms [3].

The objective of the research is implementation and quality assessment of heuristic algorithms for the MCPP.

The paper is organized as follows. First, the mathematical formulation of the problem is pointed out. The next section is dedicated to related works. In the next part a brief description of implemented algorithms and methodology of the research are presented. Then, already obtained results are revealed. In the final part, the expected results and future directions of research are described.

In this article, in accordance with generally accepted definitions, under the understanding of an understanding directed edge, under the edge is an undirected edge.

II. Mathematical formulation of the problem

The weight strongly connected mixed multigraph $G = < V, E \cup A, C >$ is given, where V is the set of multigraph's vertices, E is the multiset of edges, A is the multiset of arcs, $C\colon E \cup A \to R_{+}$– cost function giving non-negative weights of arcs and edges between vertices [5].

Let $I = \{1, 2, \dots, |E + A|\}$, $L = \{1, 2, \dots, |V|\}$.

Indexation on the set of vertices V is defined as $inv\colon V \to L$, $\forall v_i \in V\ \forall v_j \in V\ v_i \neq v_j \Rightarrow i \neq j$, $i = inv(v_i)$. On the multiset $E \cup A$ indexation is defined as $inea\colon \mathrm{E} \cup \mathrm{A} \to I, \forall e_i \in (\mathrm{E} \cup \mathrm{A})\ \forall e_j \in (\mathrm{E} \cup \mathrm{A})\quad e_i \neq e_j \Rightarrow i \neq j, i = inea(e_i)$.

Route $\mu = (e_{p_1}, e_{p_2}, \dots, e_{p_k})$ is a solution of the MCPP that satisfies the following properties.

- $v^{-}(e_{p_1}) = v^{+}(e_{p_k})$,
  $\forall i \in \{1, 2, \dots, k-1\}\ v^{+}(e_{p_i}) = v^{-}(e_{p_{i+1}})$, where $v^{-}(e)$ is the start vertex of arc or edge e, $v^{+}(e)$ is the end vertices of arc or edge e.
- $\mathrm{E} \cup \mathrm{A} \setminus \{e_{p_1}, e_{p_2}, \dots, e_{p_k}\} = \emptyset$.

Let $C(\mu) = \sum_{i=1}^{k} C(e_i)$ is the cost of the route.

Let $\mathcal{M}$ be a set of solutions of the MCPP. It is necessary to find a route $\mu_0 \in \mathcal{M}$ that satisfies the following property $\forall \mu \in \mathcal{M}\ C(\mu_0) \leq \mathrm{C}(\mu)$ or $C(\mu_0) = \min_{\mu \in \mathcal{M}}(\mathrm{C}(\mu))$.

III. Related works

In the paper by G. Laporte the exact algorithm for an undirected and directed graph is covered and the formulation of the problem in the mixed graph is shown [6]. The polynomial algorithm and executable code of open and closed CPP in directed graph is presented in the work of Harold Thimbleby from University College London [3]. The T. Ralph's paper contains formulation of CPP on the mixed graph in terms of integer linear programming. However, solution of the MCPP is not given [7].

It was shown that several arc routing problems can be converted into equivalent ARP [8]. It opens up the possibility to solve arc routing problems, including the MCPP, as equivalent of another arc routing problem. In particular, the paper gives grounds to assume the MCPP can be converted into generalized traveling salesman problem (GTSP). Keld Helsgaun present the heuristic algorithm GLKH for solving GTSP [9, 10].

There are numerous papers and publications on the issue of the CPP, but no research on the issue of the MCPP in the multigraph has been found. To achieve the goal, algorithms and own methods of solving the MCPP in multigraph were modified and investigated. The method of transformation the MCPP into GTSP were implemented and tested.

IV. The implementation of algorithms

A. Solving MCPP based on solving directed CPP

Methods for solving the MCPP in multigraph are based on the transformation of the CPP in the mixed multigraph to the CPP in the directed multigraph. Then the CPP in the directed multigraph should be solved by the algorithm, according to Thimbleby [3].

The algorithm looks as follows:

- Let $d^{+}(v)$ the amount of arc incoming in vertex, $d^{-}(v)$ the amount of arc out coming from vertex.
- Define two sets of vertices $D^{+} = \{d^{+}(v) - d^{-}(v) > 0\}$ and $D^{-} = \{d^{+}(v) - d^{-}(v) < 0\}$.
- To construct optimal CPP solution in directed case should balance each vertex (make $d^{+}(v) = d^{-}(v)$), be adding extra path in multigraph, which connect vertex from $D^{-}\ to\ D^{+}$ and have the lowest possible cost.
- Then should construct Eulerian circuit. Eulerian cycle in new multigraph is a solution of CPP.

For solving the MCPP as the CPP in the directed multigraph three transformation methods from mixed multigraph to directed were implemented:

- *Replacement Edges algorithm.* (Replace each edge with a pair of oppositely directed arc);
- *Balanced algorithm.* (Find the degrees of vertices. Then we make the degrees of all vertices even, adding the shortest paths between odd vertices, oriented edges, if it possible. Then replace remaining edge with a pair of oppositely directed arc);
- *Greedy algorithm.* (Sort the list of edges in descending order of their cost. Then find the degrees of the vertices and make the degree of each vertex even, orienting the edges as much as possible. Then orient the edges of the graph in descending order of their cost, taking care that the graph remains strongly connected).

B. Solving MCPP based on solving GTSP

Another way of solving the MCPP in the multigraph is based on the GLKH algorithms [9].

The MCPP can be transforms into an equivalent ARP. When problem defined in directed multigraph, it can be transformed into asymmetric TSP. When problem defined in mixed or undirected multigraph, it can be transformed into an asymmetric GTSP. [8]

TABLE I. FORMULAS FOR COMPUTING ARC COSTS OF TRANSFORMED GRAPH

		1	2	3	4	5	6	7	8
		v_{12}^{1}	v_{13}^{1}	v_{13}^{2}	v_{21}^{1}	v_{23}^{1}	v_{23}^{2}	v_{31}^{1}	v_{32}^{1}
1	v_{12}^{1}	-	$s_{21} + c_{13}^{1}$	$s_{21} + c_{13}^{2}$	$s_{22} + c_{21}^{1}$	$s_{22} + c_{23}^{1}$	$s_{22} + c_{23}^{2}$	$s_{23} + c_{31}^{1}$	$s_{23} + c_{32}^{1}$
2	v_{13}^{1}	$s_{31} + c_{12}^{1}$	-	$s_{31} + c_{13}^{2}$	$s_{32} + c_{21}^{1}$	$s_{32} + c_{23}^{1}$	$s_{32} + c_{23}^{2}$	$s_{33} + c_{31}^{1}$	$s_{33} + c_{32}^{1}$
3	v_{13}^{2}	$s_{31} + c_{12}^{1}$	$s_{31} + c_{13}^{1}$	-	$s_{32} + c_{21}^{1}$	$s_{32} + c_{23}^{1}$	$s_{32} + c_{23}^{2}$	$s_{33} + c_{31}^{1}$	$s_{33} + c_{32}^{1}$
4	v_{21}^{1}	$s_{11} + c_{12}^{1}$	$s_{11} + c_{13}^{1}$	$s_{11} + c_{13}^{2}$	-	$s_{12} + c_{23}^{1}$	$s_{12} + c_{23}^{2}$	$s_{13} + c_{31}^{1}$	$s_{13} + c_{32}^{1}$
5	v_{23}^{1}	$s_{31} + c_{12}^{1}$	$s_{31} + c_{13}^{1}$	$s_{31} + c_{13}^{2}$	$s_{32} + c_{21}^{1}$	-	$s_{32} + c_{23}^{2}$	$s_{33} + c_{31}^{1}$	$s_{33} + c_{32}^{1}$
6	v_{23}^{2}	$s_{31} + c_{12}^{1}$	$s_{31} + c_{13}^{1}$	$s_{31} + c_{13}^{2}$	$s_{32} + c_{21}^{1}$	$s_{32} + c_{23}^{1}$	-	$s_{33} + c_{31}^{1}$	$s_{33} + c_{32}^{1}$
7	v_{31}^{1}	$s_{11} + c_{12}^{1}$	$s_{11} + c_{13}^{1}$	$s_{11} + c_{13}^{2}$	$s_{12} + c_{21}^{1}$	$s_{12} + c_{23}^{1}$	$s_{12} + c_{23}^{2}$	-	$s_{13} + c_{32}^{1}$
8	v_{31}^{2}	$s_{21} + c_{12}^{1}$	$s_{21} + c_{13}^{1}$	$s_{21} + c_{13}^{2}$	$s_{22} + c_{21}^{1}$	$s_{22} + c_{23}^{1}$	$s_{22} + c_{23}^{2}$	$s_{23} + c_{31}^{1}$	-

TABLE II. THE COST MATRIX OF TRANSFORMED GRAPH

		1	2	3	4	5	6	7	8
		v_{12}^{1}	v_{13}^{1}	v_{13}^{2}	v_{21}^{1}	v_{23}^{1}	v_{23}^{2}	v_{31}^{1}	v_{32}^{1}
1	v_{12}^{1}	-	6	7	1	2	3	6	5
2	v_{13}^{1}	5	-	10	4	5	6	4	3
3	v_{13}^{2}	5	9	-	4	5	6	4	3
4	v_{21}^{1}	1	5	6	-	3	4	7	6
5	v_{23}^{1}	5	9	10	4	-	6	4	3
6	v_{23}^{2}	5	9	10	4	5	-	4	3
7	v_{31}^{1}	1	5	6	2	3	4	-	6
8	v_{32}^{1}	2	6	7	1	2	3	6	-

The process of transformation the MCPP to GTSP is to transform the original graph $G = < V, E \cup A, C >$ into equivalent problem on complete graph $\tilde{G} = < \tilde{V}, \tilde{A}, \tilde{C} >$.

Each arc $a_{ij}^{k} \in A$ between to vertices $v_i, v_j \in V$ is represented as vertex $v_{ij}^{k} \in \tilde{V}$, which must be used in the solution at least once, where k is the serial number of parallel arc.

Each edge $e_{ij}^{k} \in \tilde{E}$ between to $v_i, v_j \in V$ is represented as two vertices $v_{ij}^{k_1}, v_{ij}^{k_2} \in \tilde{V}$, one of which must be used in the solution, another may not be used, where k is a serial number of parallel edge, k_1, k_2 are the serial numbers of parallel arcs between two vertices.

After replacing the arcs and edges in vertices, the cost from each pair of vertex $v_{ab}^{k_1}, v_{cd}^{k_2} \in \tilde{V}$ in graph $\tilde{G}$ compute, as

$\widetilde{c_{ij}} = d_{bc} + c_{cd}^{k_2}$, where d_{bc} is the shortest distance between vertices $v_b, v_c \in V$ in original multigraph G.

Then, the compete graph $\tilde{G}$ is partitioned into clusters as follows: each arc and each edge is separate clusters. The number of clusters is equals to $|A \cup E|$. The graph partitioned into clusters because edge can be traverse in two ways, for solving the MCPP any way is appropriate and the problem transforms into asymmetric GTSP.

The GTSP is a variation of the Traveling Salesman Problem in which all vertices are divided into clusters, and solution consist from one vertices from each cluster [8].

The example of original multigraph is shown on Fig. 1. Each arc and edge has the cost of traverse. Each vertex has a serial number.

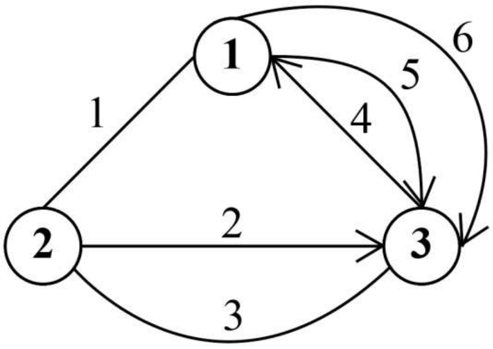


Fig. 1. Original MCPP problem

We replace each edge by a pair of two oppositely directed arcs and specify the numbering of parallel arcs between each pair of vertices (see Fig. 2). In multigraph only one arc a_{12}^1 or a_{21}^1 is required, because these arcs represent one edge. The same applies to arc a_{23}^2 or a_{32}^1.

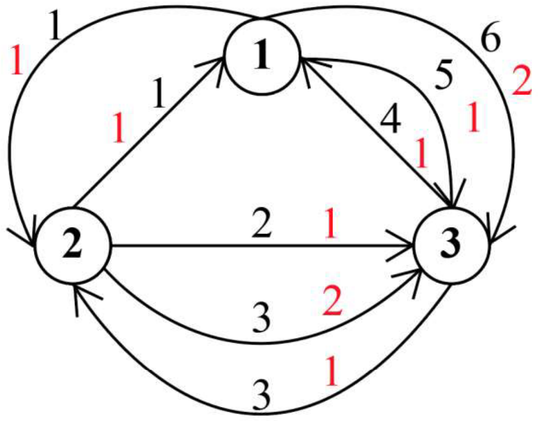


Fig. 2. The results of numbering each parallel arc

After that, should replace each arc and edge as vertex. We received new graph $\tilde{G}$ with 8 verteces. The $\tilde{V}$ can be calculates as $|\tilde{V}| = |\tilde{A}| + 2|\tilde{E}|$.

The cost from each pair of vertices is calculated by formulas (see Table 1, see Table 2). The vertices represent the edge are marked with a color in the table (different colors for different edges).

Then verteces from $\tilde{V}$ are partitioned into clusters. Fig. 3 depicts the vertices and clusters of transformed graph.

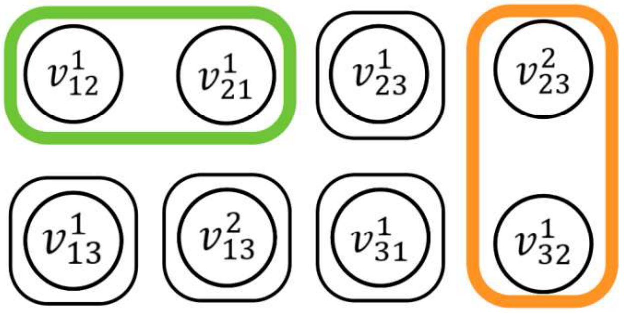


Fig. 3. The vertices and clusters of transformed problems

After transformation of original problem MCPP into GTSP, the solution for GTSP can be applied. For solving GTSP the GLKH algorithm was used [9, 10]. The algorithm transforms the GTSP problem into asymmetric TSP and then solve TSP using the LKH algorithm [10]. It was shown that GLKH's performance is good. All instances of GTSP benchmark instances were tested [11]. The maximum error rate was no more than 5%.

V. EXPERIMENTAL RESEARCH

The code for graph transformation and GLKH algorithm (used Keld Helsdgaun library) was written in C++. The code for Balanced, Greedy and Replacement Edges transformation algorithms also were written in C++. For writing algorithm for solving CPP in directed multigraph in C++ the code written in Java [3] and code written in C# [12] were used.

To measure the time characteristics and error rate of the algorithms, each algorithm was assessed as follows:

- Test data were loaded in console program. For testing error rate of solutions and computational results, the large library of test instances for MCPP in graph is used. The library contains test data for 500, 1000, 1500, 2000 and 3000 vertices. For each vertices size 24 test files are presented. Some test input sets no have exact solution. [11] For multigraph, the test data sets were not found. However, graph is a special case of multigraph (without parallel arcs and edges) and algorithm can be tested on graph data sets;
- The measurements for each input data set were carried out 10 times. The results of computational time were obtained as the average of 10 runs of the program:

$$T_{av} = \frac{T_1 + \cdots + T_{10}}{10} \tag{2}$$

- Error rate of the algorithms was evaluated according to the formula:

$$Error = \frac{C(\mu) - C(\mu_0)}{C(\mu_0)} \tag{3}$$

where $C(\mu)$ is the resulting length of the route of the MCPP using developed algorithm, $C(\mu_0)$ is the optimum length of the route of the MCPP given in input data.

All test provides on Mac Book Pro 13 retina 2014 (Intel Core i5, 2.6 GHz).

Since the algorithm GLKH is based on heuristic for solving TSP LKH, which has a lot of parameters for solution process, the first step of experiment consists of identifying the optimal parameters for solving.

TABLE III. RESULTS FOR GTSP WITH VARIES PARAMETRES

	Av. time, sec	**Av. error, %**
AC = 10, IP = 10, MC = 12, MT = 2, 3-opt	14,467	5,24%
AC = 10, IP = 10, MC = 12, MT = 2, 4-opt	15,000	6,54%
AC = 10, IP = 10, MC = 12, MT = 4, 5-opt	15,158	3,00%
AC = 10, IP = 10, MC = 2, MT = 50, 3-opt	26,079	5,35%
AC = 10, IP = 10, MC = 2, MT = 50, 4-opt	28,438	5,17%
AC = 10, IP = 10, MC = 2, MT = 200, 2-opt	38,029	4,63%
AC = 10, IP = 10, MC = 2, MT = 200, 3-opt	36,513	4,58%
AC = 10, IP = 10, MC = 2, MT = 200, 4-opt	30,717	5,00%
AC = 10, IP = 10, MC = 2, MT = 200, 5-opt	33,913	4,56%
AC = 10, IP = 10, MC = 2, MT = 1000, 2-opt	72,983	4,00%
AC = 10, IP = 10, MC = 2, MT = 1000, 3-opt	59,333	4,60%
AC = 10, IP = 10, MC = 2, MT = 1000, 4-opt	64,963	4,47%
AC = 10, IP = 10, MC = 2, MT = 1000, 5-opt	68,525	3,89%

The following parameters were assessed [13, 14]:

- ASCENT_CANDIDATE (AC). The count of edge candidate, associated with each node during the ascent. Results for 10, 100, 500 candidates were compute.
- INITIAL_PERIOD (IP). The length of first ascent. Results for 10, 100, 500, 1000 length were compute.
- MAX_CANDIDATES (MC). The maximum number of edge candidate associated with each node. Results for 2, 6, 12 candidates were compute.
- MAX_TRIALS (MT). The maximum number of trials. Results for 2, 50, 200, 1000 trials were compute.
- MOVE_TYPE (k-opt). A value $K >= 2$ signifies that a sequential $K-opt$ move is to be used. Results for 2, 3, 4, 5 k were compute.

Squeezing results for first step of experiment are representing in Tables 3.

TABLE IV. COMPUTATIONAL RESULTS FOR GTSP WITH CHOSEN PARAMETERS

	Av. time, sec	**Av. error, %**
|V|=500	171,713	0,58%
|V|=1000	500,729	0,92%
|V|=1500	836,333	1,05%
|V|=2000	1139,467	1,19%
|V|=3000	2403,892	1,20%

After analyzing the Table 3, the most appropriate parameters were chosen:

ASCENT_CANDIDATES = 500;
INITIAL_PERIOD = 100;
MAX_CANDIDATES = 15;
MAX_TRIALS = 50;
MOVE_TYPE = 5.

For chosen parameters all test sets (from $|V| = 500$ to $|V| = 3000$) were tested (see Table 4). For problem MB3065 chosen algorithm improved the existing solution. Previous tour length is 204160, founded by modification of algorithm is 203105.

As seen from Tables 3 and 4, chosen parameters entail a long computational time, which is not always appropriate.

The second step of experiments is definition of Pareto-optimal algorithms (criteria: error and computational time) for solving MCPP, evaluate GLKH algorithm with varies parameters from first step and algorithms based on solving directed CPP.

TABLE V. COMPUTATIONAL RESULTS FOR PARETO-OPTIMAL ALGORITHMS

		Av. time, sec	**Av. error, %**
GLKH with parameters AC = 500, IP = 100, MC = 20, MT = 100, 5-opt.	|V|=500	171,713	0,58%
	|V|=1000	500,729	0,92%
	|V|=1500	836,333	1,05%
	|V|=2000	1139,467	1,19%
	|V|=3000	2403,892	1,20%
GLKH with parameters AC = 10, IP = 10, MC = 12, MT = 2, 3-opt.	|V|=500	14,467	5,24%
	|V|=1000	69,355	5,89%
	|V|=1500	165,013	4,60%
	|V|=2000	298,653	5,01%
	|V|=3000	790,543	5,28%
GLKH with parameters AC = 10, IP = 10, MC = 12, MT = 2, 5-opt.	|V|=500	15,158	3,00%
	|V|=1000	78,242	3,87%
	|V|=1500	177,883	4,39%
	|V|=2000	317,171	4,54%
	|V|=3000	895,804	4,80%
Balanced	|V|=500	5,110	43,33%
	|V|=1000	12,455	63,56%
	|V|=1500	17,895	49,98%
	|V|=2000	33,435	54,79%
	|V|=3000	40,130	48,34%
Edges Replacement	|V|=500	4,983	49,61%
	|V|=1000	11,343	74,33%
	|V|=1500	15,654	54,14%
	|V|=2000	30,433	65,34%
	|V|=3000	38,339	58,75%

The Fig. 3 is representing the Pareto-optimal algorithms. As shown in diagram, despite the algorithm Replacement Edges and Balanced have high error rate, they have low computational time and are Pareto-optimal.

The following algorithms are Pareto-optimal:

- Transformation MCPP into GTSP and using GLKH with parameters AC = 10, IP = 10, MC = 12, MT = 2, 5-opt.
- Transformation MCPP into GTSP and using GLKH with parameters AC = 10, IP = 10, MC = 12, MT = 2, 3-opt.

- Transformation MCPP into GTSP and using GLKH with parameters AC = 500, IP = 100, MC = 20, MT = 100, 5-opt.
- Transformation MCPP into directed CPP by Balanced algorithm.
- Transformation MCPP into directed CPP by Edges Replacement algorithm.

In contradistinction to algorithm based on transformation MCPP into GTSP and using GLKH with parameters AC = 500, IP = 100, MC = 20, MT = 100, 5-opt all other Pareto-Optimal algorithm does not improve current existing solution, but have appropriate error rate and fast computation (see Table 5).

VI. Summary

The ways of solving MCPP in multigraph were proposed. Previously, the MCPP were solved only in a mixed graph.

As a result of the research, three modifications of Thimbleby's algorithms [3] for solving the MCPP in the multigraph based on transformation mixed multigraph into directed multigraph were implemented.

The algorithm for transforming MCPP in multigraph into an equivalent arc routing problem GTSP was developed.

The all presented algorithms were implemented and evaluated in test data sets. The Pareto-optimal algorithms were found. For problem MB3065 the solution was improved by 0.5%.

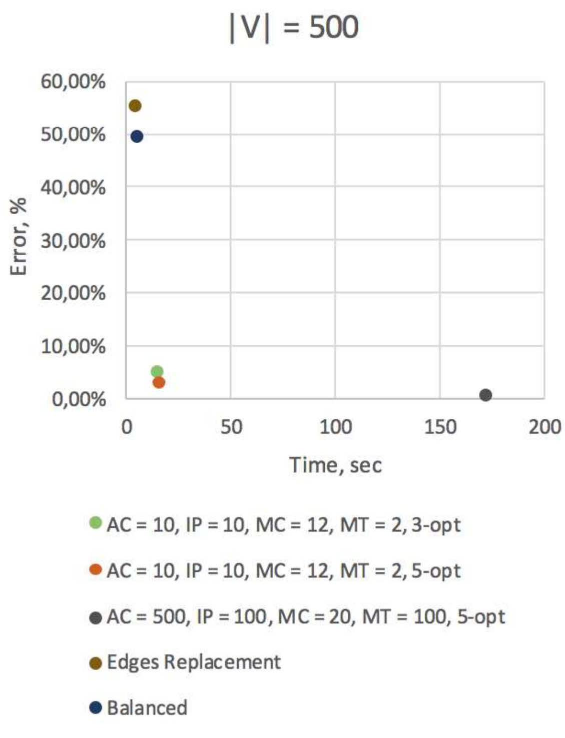


Fig. 4. Pareto-optimals MCPP algorithms

In our future work, we are going to fine-tune parameters of GLKH methods using genetic algorithms of search optimization. Further, it is possible to apply and investigated other existing algorithms for GTSP. Next, we are going to develop (prepare) test data for a mixed multigraph and to conduct experiments to ensure that a Pareto optimal group is sustainable.

References

[1] V. B. David, Decision Maths 1, London: Heinemann Educational Publishers, 2004.

[2] P. E. Vreda, "Chinese postman problem," in *Dictionary of Algorithms and Data Structures*, National Institute of Standards and Technology, 2014.

[3] H. Thimbleby, "The directed Chinese Postman Problem," *Software Practice and Experience,* vol. 33, no. 11, pp. 1081-1096, 2003.

[4] J. Edmonds, "Matching, Euler tours and the Chinese postman," *Mathematical Programming,* vol. 5, no. 1, pp. 88-124, 1973.

[5] T. K. Ralphs, "On the Mixed Chinese Postman Problem," *School of Operations Research and Industrial Engineering,,* pp. 123-127, 1993.

[6] G. Laporte, "Arc Routing Problems, Part I: The Chinese Postman Problem," *Operations Research,* vol. 43, pp. 231-242, 1993.

[7] T. K. Ralphs, "On the Mixed Chinese Postman Problem," *School of Operations Research and Industrial Engineering,* pp. 123-127, 1993.

[8] G. Blais, "Exact Solution of the Generalized Routing Problem through Graph Transformations," *Operations Research,* vol. 54, no. 8, pp. 906-910, 2003.

[9] K. Helsgaun, "Solving the Equality Generalized Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm," Roskilde University, 2014.

[10] K. Helsgaun, "Solving Arc Routing Problems Using the Lin-Kernighan-Helsgaun Algorithm," Roskilde University, 2014.

[11] Á. Corberán, "Arc Routing Problems: Data Instances," [Online]. Available: http://www.uv.es/corberan/instancias.htm. [Accessed 3 April 2017].

[12] M. Gordenko and S. Avdoshin, "ClosedCPPForDirecredAndAndirectedGraph". Patent 2014661301, 28 October 2014.

[13] K. Helsgaun, "An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic," *European Journal of Operational Research,* vol. 126, no. 1, pp. 106-130, 2000.

[14] K. Helsgaun, "LKH," [Online]. Available: http://akira.ruc.dk/~keld/research/LKH/. [Accessed 9 April 2017].

Pareto-optimal Algorithms for Metric TSP

Ekaterina N. Beresneva (Chirkova)
Faculty of Computer Science
National Research University Higher School of Economics
Moscow, Russia
enchirkova@edu.hse.ru

Scientific Advisor: Prof. Sergey Avdoshin
Software Engineering School
National Research University Higher School of Economics
Moscow, Russia
savdoshin@hse.ru

***Abstract*— The Travelling Salesman Problem (TSP) is a fundamental task in combinatorial optimization. A special case of the TSP is Metric TSP, where the triangle inequality holds. Solutions of the TSP are generally used for costs minimization, such as finding the best tour for round-the-world trip or construction of very large-scale integration schemes. Since the TSP is NP-hard, heuristic algorithms providing near optimal solutions will be considered. The objective of this article is to find a group of Pareto optimal heuristic algorithms for Metric TSP under criteria of run time efficiency and qualitative performance as a part of the experimental study. Classification of algorithms for Metric TSP is presented. Feasible heuristic algorithms and their prior estimates are described. The details of the research methodology are provided. Finally, results and prospective research are discussed.**

Keywords—travelling salesman problem, resource-efficient algorithm, heuristic algorithm, posterior estimate, computational experiment.

I. Introduction

The Travelling Salesman Problem (TSP) is one of the most widely known questions in a class of combinatorial optimization problems. Essentially, to meet a challenge of the TSP is to find a Hamiltonian circuit of minimal length. A subcase of the TSP is Metric TSP where all of the edge costs are symmetric, and they satisfy the triangle inequality.

The methods for solving the TSP have been developed for many years, and since the problem is NP-hard, it continues to be topical. The TSP has seen applications in the areas of logistics, genetics, manufacturing, telecommunications and neuroscience [1]. The most common practical interpretation of the TSP relates to the movement of people and vehicles around tours, such as searching for the shortest tour through N cities, school bus route planning, and postal delivery. In addition, the TSP plays an important role in very large-scale integration (VLSI) [2].

The purpose of this study is to determine the group of Pareto-optimal heuristic algorithms for Metric TSP by criteria of run time and qualitative performance as part of the experimental investigation.

Clearly, a study of this type is inevitably restricted by various constraints, in this research only heuristic algorithms constructing near optimal solutions in polynomial time will be considered instead of the exact ones.

The paper is structured as follows. First, the theoretical basis is described. It presents the mathematical formulation of Metric TSP, the specification of metric types and, at last, definition of Pareto optimization. Next, a classification of algorithms for Metric TSP is given, including a literature review of popular heuristics. Then the description of methods to be used is provided with their prior estimates. After that the details of the research methodology and expected results are mentioned.

II. Theoretical Basis

A. Problem Formulation

In this paper, mathematical formulation of Metric TSP is adopted as follows.

Given a complete weighted undirected graph $G = (V, V^2)$ which contains $N = |V|$ vertices. Graph vertices are indexed as $index = V \to I, I = \{1, 2, \ldots, N\}$, and $(\forall v_i \in V)(\forall v_j \in V)$ it is true that if $v_i \neq v_j$ then $i \neq j$, where $i = index(v_i)$. The distance between two vertices v_i and v_j is calculated by distance function $d(v_i, v_j)$. Here a real-valued function $d(\cdot,\cdot)$ on $V \times V$ satisfies [3]:

1. $d(v_i, v_j) \geq 0$ (non-negativity axiom)
2. $d(v_i, v_j) = 0$ if and only if $v_i = v_j$ (identity axiom)
3. $d(v_i, v_j) = d(v_j, v_i)$ (symmetry axiom)
4. $d(v_i, v_k) \leq d(v_i, v_j) + d(v_j, v_k)$ (triangle inequality axiom)

Let S be a set of all Hamiltonian cycles of G. It is defined as $S = \{p: V \to V | (p(1) = 1 \& (\forall i \in V)(\forall j \in V)$ $(p(i) = p(j) => i = j\}$. An example of a Hamiltonian circuit $s \in S$ is $(p_1, p_2, \ldots, p_N)$, where p_i is used as abbreviated notation of $p(i)$.

Weight of a Hamiltonian cycle $s \in S$ can be found according to the formula (1):

$$f(s) = d(p_1, p_N) + \sum_{i=1}^{N-1} d(p_i, p_{i+1}) \quad (1)$$

The set of vertices V is determined in Euclidean space R^n by their coordinates. Under these circumstances, the Minkowski distance of order p, where $p \geq 1$, between two vertices $v = (x_1, x_2, \ldots, x_n)$ and $w = (x_1, x_2, \ldots, x_n)$ is

defined as (2):

$$d(v, w) = \sqrt[p]{\sum_{i=1}^{n} |x_i(v) - x_i(w)|^p} \quad (2)$$

Three most used practical metrics are based on Minkowski distance. The first one is the taxicab metric, which is also known as the L_1 distance or the Manhattan distance (for $p = 1$) (3). The second is the Euclidean metric or the L_2 distance (for $p = 2$) (4). The third metric is the Chebyshev distance or the L_∞ metric (for $p \to \infty$) (5).

$$d_0(v, w) = \sum_{i=1}^{n} |x_i(v) - x_i(w)| \quad (3)$$

$$d_1(v, w) = \sqrt{\sum_{i=1}^{n} \big(x_i(v) - x_i(w)\big)^2} \quad (4)$$

$$d_2(v, w) = \max_{i=1,\dots,n} |x_i(v) - x_i(w)| \quad (5)$$

The formulation for the metric travelling salesman problem is to find such s_0 that $f(s_0) = \min_{s \in S} f(s)$ for the given metric $d \in \{d_0, d_1, d_2\}$.

B. *Resourse-efficient parameters*

Let M be a set of given heuristic algorithms for Metric TSP. There are two parameters of resource-efficiency for $m \in M$ for each N:

- $f_\varepsilon(m, N)$ – qualitative performance;
- $f_t(m, N)$ – running time.

Qualitative performance can be calculated using (7):

$$f_\varepsilon(m, N) = \frac{f(s) - f(s_0)}{f(s_0)} * 100\%, \quad (7)$$

where $f(s)$ is the obtained tour length and $f(s_0)$ is the optimal tour length.

C. *Pareto-optimality*

The algorithm $m_0 \in M$ is Pareto optimal if $(\forall m \in M)$ $\Big((m \neq m_0) \Rightarrow \big(f_\varepsilon(m) > f_\varepsilon(m_0)\big) \vee \big(f_t(m) > f_t(m_0)\big)\Big)$.

III. Literature Review

Algorithms for solving the TSP may be divided into two classes:

- Exact algorithms, and
- Heuristic (or approximate) algorithms.

Exact algorithms are aimed at finding optimal solutions. Both widely known subtypes of exact methods – linear programming and branch-and-bound techniques – are described in details by Applegate [1]. However, a major drawback is connected with their time efficiency. It is a common knowledge that there are no exact algorithms running in polynomial time. Thus, only small datasets can be solved in reasonable time. For example, the 4410-vertex problem is believed to be the largest Metric TSP ever solved with respect to optimality [4].

In this paper, only a class of heuristic search algorithms will be taken into account. They are designed to run quickly and to get an approximate solution to a given problem. Heuristic algorithms are subdivided into two groups.

The first group includes *tour construction algorithms* that have one feature in common – the tour is built by adding a new vertex at each step. The most common methods are Nearest Neighbour, Double Nearest Neighbour, and Greedy algorithms, that are represented in Flood's article [5]. Other constructive methods – insertion heuristics – are described by Johnson and McGeoch [6]. Other well-known algorithms based on minimum spanning tree are introduced by Christofides [7] [8].

The second group consists of *local-search algorithms* that have their roots in TSP papers from the 1950s [9] [10]. According to Applegate, '*... These heuristics take as input an approximate solution to a problem and attempt to iteratively improve it by moving to new solutions that are close to the original'* [1]. Most of these algorithms are described by Aarts and Lenstra [11]. Currently, the main local-search heuristic used in practice is 2-Opt heuristic [12]. It was introduced and described by Flood [5], Croes [10] and Bock [9]. The later algorithm of Lin and Kernighan [13] appeared on the basis of k-Opt tour-finding approach.

IV. Algorithms

In this paper, the following methods for solving Metric TSP will be implemented and assessed through experiments.

1) *Nearest Neighbour (NN).*

The key to NN is to initially choose a random vertex and to add repeatedly the nearest vertex to the last appended, unless all vertices are used [5].

2) *Double Ended Nearest Neighbour (DENN).*

This algorithm is a modification of NN. Unlike NN, not only the last appended vertex is taken into consideration, so the closest vertex to both of endpoints in the tour is added [5].

3) *Greedy (GRD).*

The Greedy heuristic constructs a path by adding the shortest edge to the tour until a cycle with A edges, $A < N$, is created, or the degree of any vertex exceeds two [14].

4) *Nearest Insertion (NI), Cheapest Insertion (CI), Nearest Segment Insertion (NSI).*

The fundamental idea of NI, CI and NSI is to start with an initial subtour made of the shortest edge and to add repeatedly other vertices using various rules. Depending on the algorithm the vertex not yet in the cycle should be inserted so that:

a) In NI it is the closest to any node in the tour;
b) In CI its addition to the tour gives a minor increment of its length;
c) In NSI distance between the node and any edge in the tour is minimal.

The previous step should be repeated until all vertices are added to the cycle [11].

5) Nearest Insertion Modified (NIM), Cheapest Insertion Modified (CIM), Nearest Segment Insertion Modified (NSIM).

Algorithms NIM, CIM and NSIM are variations of NI, CI and NSI respectively. The feature of modified methods is additional computation that selects the best place for each inserting node.

6) Double Minimum Spanning Tree (DMST).

DMST method is based on the construction of a minimal spanning tree (MST) from the set of all vertices. After MST is built, the edges are doubled in order to obtain an Eulerian cycle, containing each vertex at least once. Finally, a Hamiltonian circuit is made from an Eulerian circuit by sequential (or greedy) removing occurrences of each node [8].

7) Double Minimum Spanning Tree Modified (DMST-M).

This algorithm is a modification of DMST. Unlike DMST, it is necessary to remove duplicate nodes from Eulerian cycle using triangle inequality instead of greedy method.

8) Christofides (CHR).

This method is a modification of DMST that was proposed by Christofides [7]. The difference between CHR and DMST is addition of minimum weight matching calculation to the first algorithm.

9) Moore Curve (MC).

This is recursive geometric method. Vertices are sorted by the order they are located on the plane. Only the two-dimensional example of Moore curve is implemented. Figure 2 shows the order of the cells after one, two and three subdivision steps respectively [15] .

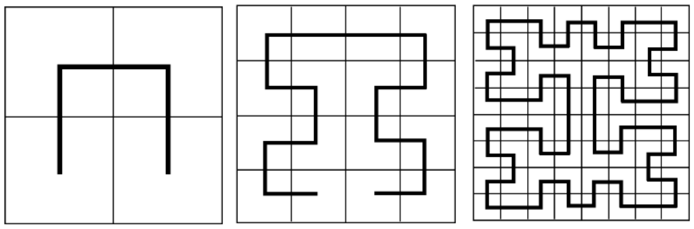


Fig. 1. The order for the Moore curve after 1, 2 and 3 subdivision steps.

10) 2-Opt.

The main idea behind 2-Opt is to take a tour that has one or more self-intersections and to remove them repeatedly. In mathematical terms, edges ab and cd should be deleted and new edges ac and bd should be inserted, if $d(a,b) + d(c,d) > d(a,c) + d(b,d)$ (Fig. 1) [11].

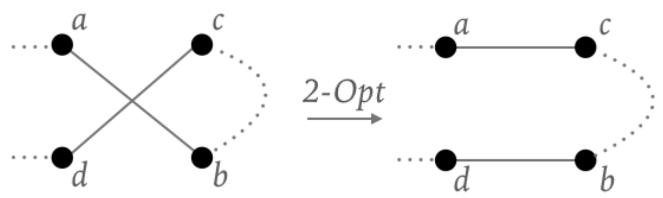


Fig. 2. 2-Opt modification.

11) Lin and Kernighan Heuristic (LKH).

LKH uses the principle of 2-Opt algorithm and generalizes it. In this heuristic, the k-Opt, where $k = \overline{2..\sqrt{N}}$, is applied, so the switches of two or more edges are made in order to improve the tour. This method is adaptive, so decision about how many edges should be replaced is taken at each step [13].

Estimated upper bounds for the algorithms can be calculated as are the ratio of $\frac{f(s)}{f(s_0)}$ (see Table 1). According to [16], for any k-Opt algorithm, where $k \leq N/4$, problems may be constructed such that the error is almost 100%. So 2-Opt and LKH algorithms have approximate upper bound 2.

TABLE I. UPPER-BOUND ESTIMATES OF ALGORITHMS

#	Algorithm	Upper-bound estimate
1	NN DENN	$0.5\lceil\log_2 N + 1\rceil$
2	GRD	$0.5\lceil\log_2 N + 1\rceil$
3	NI, CI, NSI NIM, CIM, NSIM	$2 - \frac{2}{N}$
4	DMST, DMST-M	$2 - \frac{2}{N}$
5	CHR	$\frac{3}{2}$
6	2-Opt	≈ 2
7	LKH	≈ 2
8	MC	$\log N$

Running time estimates of the algorithms are represented in Table 2.

TABLE II. RUNNING TIME OF ALGORITHMS

#	Algorithm	Running time
9	NN DENN	$O(N^2)$
10	GRD	$O(N^2 \log N)$
11	NI, CI, NSI NIM, CIM, NSIM	$O(N^2)$
12	DMST, DMST-M	$O(N^2)$
13	CHR	$O(N^3)$
14	2-Opt	$O(N^2)$
15	LKH	$O(N^{2,2})$
16	MC	$O(N \log N)$

V. EXPERIMENTAL RESEARCH

This section documents details of the research methodology. The experiment is carried out on a 1.3 GHz Intel Core i5 MacBook Air. It includes the qualitative performance and the run time efficiency of the current implementations.

Heuristics are implemented in C++. VLSI data sets from an open library TSPLIB [2] are selected as input data for algorithms. There are 102 instances in the VLSI collection that range in size from 131 vertices up to 744,710 vertices. There is one data set for each number of vertices. The integer Euclidean metric distance is used, so coordinates of nodes and distances between them have integer values, thus without loss of generality (4) is transformed into:

$$d_1(v,w) = \left\lfloor \sqrt{|x(v) - x(w)|^2 + |y(v) - y(w)|^2} + 0.5 \right\rfloor$$

The computational experiment corresponds to the following scenario:

```
for algorithm $m$ in range [1…9, 11] (see IV)
   for data set $N$ in range [1…102]
      for i in range [1…11]
         $f_{\varepsilon_i}(m,N)$, $f_{t_i}(m,N)$ are calculated
            if i > 1 then
               $f_{\varepsilon_{min}}(m,N)$ is memorized
               $f_{t_{sum}}(m,N)$ is calculated
      $f_{t_{avg}}(m,N) = \frac{f_{t_{sum}}(m,N)}{10}$
      // 2-Opt stage
      if $m != 11$ ($m$ is not LKH)
         $f_\varepsilon(m+2\text{–}Opt,N)$, $f_t(m+2\text{–}Opt,N)$ are
         calculated
   $E\left(f_{\varepsilon_{min}}(m,N)\right)$ for all $N$ is calculated
   $\sigma\left(f_{\varepsilon_{min}}(m,N)\right)$ for all $N$ is calculated
   $max\left(f_{\varepsilon_{min}}(m,N)\right)$ for all $N$ is calculated
   $min\left(f_{\varepsilon_{min}}(m,N)\right)$ for all $N$ is calculated
```

Metrics used in scenario have following meanings:

- $f_{\varepsilon_{min}}(m,N)$ – best qualitative performance of m,
- $f_{t_{sum}}(m,N)$ – accumulative running time of m (sec),
- $f_{t_{avg}}(m,N)$ – average running time of m (sec),
- $f_\varepsilon(m+2\text{–}Opt,N)$ – qualitative performance of $m+2\text{–}Opt$,
- $f_t(m+2\text{–}Opt,N)$ – running time of $m+2\text{–}Opt$,
- $E\left(f_{\varepsilon_{min}}(m,N)\right)$ – expected value of qualitative performance of m for all N,
- $\sigma\left(f_{\varepsilon_{min}}(m,N)\right)$ – standard deviation of qualitative performance of m for all N,
- $max\left(f_{\varepsilon_{min}}(m,N)\right)$ – maximum value of qualitative performance of m for all N,
- $min\left(f_{\varepsilon_{min}}(m,N)\right)$ – minimum value of qualitative performance of m for all N.

Qualitative performance metrics are represented in Table 3. Table color scheme varies from green (the best result in a column) to red (the worst value in a column).

VI. Pareto-optimal algorithms

We decided to select six different data sets with N = {1084, 5087, 10150, 30440, 52057, 104814} to plot charts that illustrate Pareto-optimal algorithms.

The chart with N = 10150 is shown below (see Figure 3). The horizontal axis represents the time performance of methods in seconds. The vertical axis shows the gap between optimal and obtained solutions, expressed in percent. Pareto-optimal methods are highlighted in red. The points which are represented by Pareto solutions are bigger than non-Pareto-optimal solutions.

TABLE III. Qualitative performance metrics of algorithms

Algorithm	E(ε)	σ(ε)	max ε	min ε
LKH 5/10	0,07%	0,05%	0,23%	0,00%
CHR + 2-Opt	5,77%	0,68%	11,02%	3,47%
GRD + 2-opt	6,22%	0,70%	9,89%	4,69%
DENN + 2-opt	10,95%	4,42%	23,51%	3,82%
NN + 2-opt	11,44%	1,87%	24,77%	4,26%
CHR	12,61%	1,08%	17,79%	9,31%
CIM + 2-opt	13,05%	2,29%	21,86%	6,74%
NIM + 2-opt	14,60%	5,53%	29,66%	5,86%
DMST-M + 2-opt	16,08%	8,39%	40,61%	4,80%
NSIM + 2-opt	17,63%	5,82%	33,65%	8,92%
GRD	18,12%	2,91%	31,34%	10,30%
DMST + 2-opt	19,08%	9,54%	39,12%	6,91%
CIM	20,31%	1,44%	27,54%	12,46%
DENN	23,28%	1,62%	32,53%	13,88%
NN	23,94%	1,64%	30,97%	12,94%
CI	26,25%	2,70%	33,05%	17,94%
NIM	27,98%	1,89%	35,29%	14,89%
DMST-M	32,41%	3,15%	41,68%	18,55%
MC + 2-opt	32,41%	22,54%	177,83%	6,21%
NSIM	36,23%	4,23%	48,17%	19,15%
DMST	40,09%	2,34%	48,88%	33,16%
NSI	43,55%	5,64%	55,61%	25,89%
NI	52,46%	3,19%	60,94%	36,77%
MC	63,93%	25,58%	242,41%	33,07%

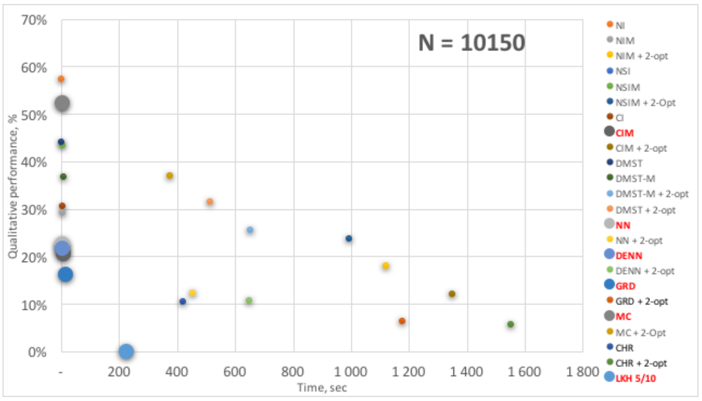


Fig. 3. Pareto-optimal algorithms, N = 10150.

Full results are presented in Table 4. Dash sign is used because some methods have not tested on large N yet. So we cannot claim whether these algorithms are Pareto-optimal or not. There are three heuristics included in each Pareto-optimal group. They are CIM, NN and MC.

TABLE IV. Pareto-optimal algorithms for 6 groups of N

N = 1084	N = 5087	N = 10150	N = 30440	N = 52057	N = 104814
CI	CI				
CIM	CIM	CIM	CIM	CIM	CIM
				CIM+2-Opt	CIM+2-Opt
NN	NN	NN	NN	NN	NN
		DENN	DENN	DENN	DENN
			DENN+ 2-Opt		DENN+ 2-Opt
		GRD			
MC	MC	MC	MC	MC	MC
CHR				-	-
LKH	LKH	LKH	LKH	-	-

However, it should be noted that despite the lack of information on LKH algorithm for N = 52057 and N = 104814, this method continues to be the most perspective heuristic. As it can be seen from Table 3, expected value of qualitative performance of LKH is 0.07%. From there, we decided to add LKH to the group of Pareto-optimal algorithms.

Overall, the final group of Pareto-optimal algorithms consists of **CIM**, **NN**, **MC** and **LKH** heuristics.

VII. Conclusion

The presented study is undertaken to determine what heuristics for Metric TSP should be used in specific circumstances with limited resources.

This paper provides an overview of some heuristic algorithms implemented in C++ and tested on the VLSI data set. In the course of computational experiments, the comparative figures are obtained and on their basis multi-objective optimization is provided. Overall, the final group of Pareto-optimal algorithms consists of CIM, NN, MC and LKH heuristics.

In our future work, we are going to fine-tune parameters of CHR and LKH methods using genetic algorithms of search optimization. Further, it is possible to increase the number of heuristic algorithms, to transit to other types of test data and to conduct experiments using metrics d_0, d_2 in order to ensure that a Pareto optimal group is sustainable.

The practical applicability of our findings is to present Pareto optimal algorithms that lead to solutions with maximum accuracy under the given resource limitations. The results can be used for scientific purposes by other researchers and for cost minimization tasks.

References

[1] D. L. Applegate, The Traveling Salesman Problem, Princeton: Princeton University Press, 2006.

[2] Heidelberg University, "TSPLIB," [Online]. Available: https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. [Accessed 01 02 2017].

[3] Wikipedia, "Multi-objective optimization," [Online]. Available: https://en.wikipedia.org/wiki/Multi-objective_optimization. [Accessed 03 02 2017].

[4] University of Waterloo, "Status of VLSI Data Sets," [Online]. Available: http://www.math.uwaterloo.ca/tsp/vlsi/summary.html. [Accessed 08 02 2017].

[5] M. M. Flood, "The traveling-salesman problem," *Operation research*, vol. 4, pp. 61-75, 1956.

[6] D. Johnson and L. McGeoch, "The Traveling Salesman Problem: A Case Study," in *Local Search in Combinatorial Optimization*, Chichester, 1997, pp. 215-310.

[7] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, CMU, 1976.

[8] N. Christofides, Graph theory - An Algorithmic Approach, New York: Academic Press, 1974.

[9] F. Bock, "An algorithm for solving "traveling-salesman" and related network optimization problems," in *Unpublished manuscript a-ssociated with talk presented at the 14th ORSA National Meeting*, 1958.

[10] G. Croes, "A method for solving travelling salesman problems," *Operation Resources*, vol. 6, pp. 791-812, 1958.

[11] E. Aarts and J. K. Lenstra, Local Search in Combinatorial Optimization, Princeton, New Jersey: Princeton University Press, 2003.

[12] G. Gutin and A. Punnen, The Traveling Salesman Problem and Its Variations, vol. 12, Kluwer, Dordrecht: Springer US, 2002.

[13] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *EJOR*, vol. 12, pp. 106-130, 2000.

[14] W. J. Cook, Combinatorial optimization, New York: Wiley, 1998.

[15] K. Buchin, "Space-Filling Curves," in *Organizing Points Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes*, Berlin, Free University of Berlin, 2007, pp. 5-30.

[16] D. E. Rosenkrantz, R. E. Stearns and P. M. Lewis II, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput*, p. 563–581, 1977.

Rapid Prototyping of Smart Mirror for Smart Home Environment

Sergey Smetanin
Faculty of Business and Management
National Research University Higher School of Economics
Moscow, Russia
sismetanin@gmail.com

***Abstract*— In the last few years, the Internet of Things (IoT) tends to get a significant attention both from the academic and the industrial spheres. Consequently, it provoked a rapid development of various projects for home automation in general and smart devices, particularly. This paper aims at discussing the concept of the smart mirror prototype. The basic features are proposed and described in details. As a result, the prototype of the smart mirror is expected to be implemented in this research. In further work, the additional augmentation functional might be added.**

Keywords—smart mirror; smart home, smart display.

I. INTRODUCTION

In the last few years, the academic and the industrial communities draw their attention to the Internet of Things (IoT). Consequently, it provoked a rapid development of various projects for home automation. Smart mirrors come from the idea of bringing technology into everyday objects: the traditional mirrors. A smart mirror is a device that functions as a mirror with the additional capability of both displaying multimedia data and interacting directly with the user.

This paper describes the functional and the structure of the smart mirror prototype, which is powered by an Android device. The aim of this paper is to specify the process of smart mirror development and to propose the basic functional of the smart mirror, which will be implemented in further work.

The paper is organized as follows. In section 2 the review of literature is provided. The prototype of the smart mirror is described in section 3. Section 4 is focused on the key smart mirror features. Section 5 states the results of the paper.

II. RELATED WORK

The smart environment systems are capable not only to perform routine tasks but also to positive affect to the user's feelings. The FitMirror project [1] aims at helping people with serious problems to get up after sleeping and to get motivated through the day using activity tracking, emotions recognition, and interactive games. This smart mirror consists of a monitor with spy-foil, a Microsoft Kinect v2, and a Wii Balance Board.

The Virtuoso project [10] tends to create a seamless interactive support for fitness and wellness activities in touristic resorts. The general idea is to evaluate the current physical state of the user through an interactive mirror. Considerable attention is paid to such physiological parameters as fat and glossary balance, glucose metabolism, and circulation of blood. In addition, the discussion of mirror setup is provided to validate the Virtuoso results in a holiday resort scenario.

The paper [13] propose a novel interface to support smart home technologies. The smart mirror display is made by using half mirror film attached to a general display like a monitor. Speech recognition and face recognition are implemented to operate the displayed content.

HomeMirror [8] is one of the most popular open-source smart mirror project, which is designed as an Android application powering the mirror. Using API services by Yahoo Finance, Forecast.io, the BBC, and XKCD, it allows users to obtain different kinds of information, e.g. calendar information, weather forecast, news, and stock price swings. Similar open-sourced projects [3], [9], and [11] also implement the basic smart mirror functional.

Regardless of the rapid development in the academic and the industrial communities, the features of these mirrors are limited. Moreover, the embedded functions are significantly vary depending on the applied sphere of smart mirror usage. The basic features for the smart mirror are proposed in the next section.

III. THE PROTOTYPE

For this project a two-way mirror is required, that is, the special panel, which allows the light to pass from the rear outwards, and also reflect the light off the front, depending on the light condition on the other side [4]. If the other side is bright, the two-way mirror will be transparent like a normal glass. Otherwise, only the reflected image will be shown. Thus, if a screen will be put on the other side of the two-way mirror, the reflected image will be shown in the dark parts of the screen, while the bright parts of the screen will be overlaid to the smart mirror. Covering the whole area of the mirror seems to be an ideal option, because it will be possible to cover it all with overlaid multimedia information. In the context of prototyping, it was decided to use an Android tablet as a hardware part of the smart mirror.

The software part of the smart mirror will be implemented as a native Android application using Java programming language and Android SDK. Android Studio was selected as

IDE because of providing high-quality tools for building applications on every type of Android device.

This prototype could also be integrated into a real smart home concept [2], [7], [12] by connecting the mirror with other components. The basic features of the smart mirror are proposed in the following section.

IV. Features

The overall idea of the smart mirror is both to display multimedia data and to interact directly with the user. Firstly, it should provide with general information such as time, weather, and news. Secondly, the user identification features should be implemented to support multi-user interaction and personalized content. Thirdly, the personal calendar data should be displayed for each identified user. Fourthly, the intelligent personal assistant should be integrated. By the end, a few additional features related to food delivery services and streaming services could be implemented. The detail description of each feature is discussed below.

A. *Information dashboard*

The default smart mirror screen should display the dashboard, which provides current time, news, and weather information. News and weather forecast could be obtained via public API of such services as Google News and Dark Sky respectively.

B. *User identification*

User identification is the first step to multi-user interaction and personalized content. A face recognition or voice recognition based approaches could be used to identify users and unlock their personal profiles to get access to private data and applications.

C. *Personal calendar*

After the user recognition, the information from the online time-management and the scheduling calendar service might be presented at the mirror. The Google Calendar API provides developers with an ability to find and view public calendar events for unauthorized users and to access and modify private calendars and events on those calendars for authorized users.

D. *Intelligent Personal Assistant*

Intelligent Personal Assistants (IPAs) such as Apple’s Siri, Google’s Google Now, and Microsoft’s Cortana are emerging as one of the fastest growing Internet services [5]. They have an ability to access the date from a variety of sources and to organize and maintain this information. In this project, personal assistant reminders from Google Now will be displayed at the mirror.

E. *Additional functional*

According to the online food ordering report [6] from the Cornell University, well-designed self-service ordering systems provide customers with substantial control over the pace of their transaction and allow them to limit the amount of personal interaction they experience [6], resulting in increased satisfaction from the ordering process. The smart mirror could be integrated with the international UberEATS service to allow users to order food delivery.

The proliferation of the Internet, the increased speed of Internet connections facilitated the rapid propagation of media streaming services usage. Due to the presence of the screen, the smart mirror could implement the media player's functions based on content from streaming services, e.g. Netflix, Amazon Video, HBO Now, etc.

V. Results

As a result, the prototype of the Android powered smart mirror for home automation is expected to be implemented. The key features of the prototype were specified and described in details. The future research will be focused on the integration with smart home systems. Moreover, the additional augmentation features could be implemented.

References

[1] D. Besserer, J. Bäurle, A. Nikic, F. Honold, F. Schüssel, and M. Weber, “Fitmirror: a smart mirror for positive affect in everyday user morning routines,” *Proceedings of the Workshop on Multimodal Analyses enabling Artificial Agents in Human-Machine Interaction - MA3HMI '16*, 2016.

[2] G. Chong, L. Zhihao, and Y. Yifeng, “The research and implement of smart home system based on Internet of Things,” *2011 International Conference on Electronics, Communications and Control (ICECC)*, 2011.

[3] E. Cohen, “evancohen/smart-mirror,” *GitHub*, 06-Sep-2015. [Online]. Available: https://github.com/evancohen/smart-mirror. [Accessed: 28-Mar-2017].

[4] D. B. S. Craftsman, “Mirror, Smart Mirror on the wall...,” *The Labs | Novoda*, 12-Jul-2016. [Online]. Available: https://www.novoda.com/blog/mirror-smart-mirror-on-the-wall/. [Accessed: 29-Mar-2017].

[5] J. Hauswald, L. Tang, J. Mars, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, and V. Petrucci, “Sirius: An Open End-to-End Voice and Vision Personal Assistant and Its Implications for Future Warehouse Scale Computers,” *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '15*, 2015.

[6] S. E. Kimes, “The Current State of Online Food Ordering in the U.S. Restaurant Industry,” Cornell University, Ithaca, New York, rep., 2011.

[7] S. Kumar, “Ubiquitous Smart Home System Using Android Application,” *International journal of Computer Networks & Communications*, vol. 6, no. 1, pp. 33–43, 2014.

[8] H. M. Mittelstaedt, “HannahMitt/HomeMirror,” *GitHub*, 20-May-2016. [Online]. Available: https://github.com/HannahMitt/HomeMirror. [Accessed: 28-Mar-2017].

[9] S. Monnerat, “Shinao/SmartMirror,” *GitHub*, 21-Feb-2016. [Online]. Available: https://github.com/Shinao/SmartMirror. [Accessed: 28-Mar-2017].

[10] M. Saba, R. Scateni, F. Sorrentino, L. D. Spano, S. Colantonio, D. Giorgi, M. Magrini, O. Salvetti, N. Buonaccorsi, and I. Vitali, “Smart Mirror Where I Stand, Who Is the Leanest in the Sand?,” *Universal Access in Human-Computer Interaction. Access to Learning, Health and Well-Being Lecture Notes in Computer Science*, pp. 364–373, 2015.

[11] M. M. Teeuw, “MichMich/MagicMirror,” *GitHub*, 16-Feb-2014. [Online]. Available: https://github.com/MichMich/MagicMirror. [Accessed: 28-Mar-2017].

[12] J. Wu, L. Huang, D. Wang, and F. Shen, “R-OSGi-based architecture of distributed smart home system,” *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1166–1172, 2008.

[13] M.-C. Youn and S. Jeong, “Efficient Supporting Method Based on Smart Mirror for Smart Home System,” *Advanced Science and Technology Letters*, vol. 141, pp. 169–172, 2016.

Inverse kinematics in ultralight UAV control problem with additional on-board microcomputer.

Vasiliy Kaliteevskiy
Saint-Petersburg State University
Email: vkalit@gmail.com

Konstantin Amelin
Saint-Petersburg State University
Email: konstantinamelin@gmail.com

***Abstract*—The possibility of using additional on-board microcomputer for collaboration with autopilot to increase functionality of UAV is considered. Using the microcomputer TRIK is proposed. The dinamic and kinematic of flight process are described. Several coordinate systems for describe the dynamic behaviors of UAV are given. The architecture of the autopilot control unit for the UAV is considered. The method of partly implement the autopilot systems control modules to microcomputer is proposed. Simulation results of motions model in Simulink are described. Testing the proposed module on the real UAV is planned.**

I. Introduction

Unmanned aerial vehicles play an increasingly prominent role in military defensive purposes. UAV successfully solve the problem of reconnaissance , surveillance, and communications, even being in extreme conditions and long flights.

Recently, UAVs have also been successfully used to solve many civilian tasks, covering a wide range of possibilities, such as the work of emergency services, border patrols, monitoring of agricultural crops forestry and fisheries control, also mapping, monitoring of green areas, monitoring of oil and gas facilities and many others.

There are a lot of different UAVs in the world, differing in their specifications and set of characteristics (purpose, weight, size, flight duration and flight altitude, launch and landing system, autopilot and navigation systems, aerial and video format, etc.) [1].

For the successful implementation of the above tasks have to choose the right components for the inner and outer stuffing UAV, leaving without due attention to the software. Thus, it is necessary to maximize the UAV's own capabilities, which includes both the selection of high-quality hardware solutions designed for possible extreme conditions and increased loads, and the software that provides stable uninterrupted control of the UAV in conditions of noisy and delays from the sensors. The control module should also be optimized in terms of power consumption, since the UAVs have a limited supply of energy due to the weight of the batteries, which affects the flight performance [2].

The subject of this paper is the description and development of the UAV control chain from receiving data from sensors to setting a signal for control mechanisms in conditions of noisy and delayed. It also requires the selection of hardware and the practical implementation of such control [1].

The problem is that from the sensor readings to the decision making by the UAV control unit, a large number of transformations and calculations must be performed, and in order for the control unit to work effectively in real time, these calculations must be carried out in an optimal way [2].

In the design of UAV flight physics, a number of problems arise. First, the UAV model can and must be viewed in different coordinate systems, which makes it necessary to constantly use spatial mathematical transformations. Secondly, the various forces and moments that act on the drone are also described in the UAV system, but the sensors available to the UAV only partially give evidence to this system. Thirdly, a non-trivial task is to describe inverse kinematics of the flight process. To describe the motion of a UAV with six degrees of freedom, it will take 12 different variables describing the coordinates, velocity, angles and angular moments of the UAV in the framework of nonlinear differential equations describing the physics of UAV flight.

It is also necessary to consider all the forces and moments that act on the UAV at the time of movement. It should be noted that this includes the wind, which plays a very important role in the case of unmanned aerial vehicles. At what comes here as a constant wind, available in some place for a while, and some gusts of wind that need to be taken into account. It is also worth noting that the forces and moments of the forces acting on the UAV are highly dependent on the characteristics of the UAV itself, which must also be taken into account. This is both the surface area and shape of the wing, as well as the fact of the presence or absence of the UAV plumage.

For the efficiency of the control module, the linearization of the differential equations is performed. Thus, all the forces and moments acting on the UAV will be considered as longitudinal and lateral, which greatly simplifies the understanding of the flight process, as well as interaction with it with the help of an autopilot.

As the final part, we consider preparation the hardware component, which was chosen as a microcontroller TRIK together with the autopilot ArduPilot. But before we begin implementation on a real microcontroller, the system must be tested. For this purpose, the Simulink simulation environment is used in the Matlab application package.

The aim of this work is to study the physics of the flight process, to simulate the flight process and to get the software and hardware solution for automatic piloting of the UAV.

II. ArduPilot flight controller

The popular used solution for autopiloting the UAV is the ArduPilot flight controller [3]. This product is a full-fledged UAV solution that allows, in addition to radio-controlled remote piloting, automatic control over a previously created route, that is, flight by points, and also has a two-way transfer of telemetry data from a board to a railway station. It is developed by the DIY Drones community and is based on the open source project Arduino .

Autopilot has good advantages [3]:

1) Low price.
2) Ability to set up to 166 flight points.
3) Editing a route in flight.
4) Wireless configuration settings.
5) Support for various frames and forms of the UAV.
6) Aircraft simulator support via Mission Planner software.

III. Proposed solution

A. Reference frames

It is proposed to implement its own autopilot in order to obtain customizable, scalable software, through which it is possible to further implement various innovative solutions, as well as test various mathematical ideas and hypotheses.

Before embarking on the physics of flight, including the basic set of forces and moments acting on the UAV, it is necessary to specify various necessary frames of reference, which will be used later in the calculation of the physics of flight. This is due to a number of reasons:

1) The classical equation of Newton's motion is described in a fixed, inertial frame of reference. However, it is easier to describe in the UAV reference system.
2) The aerodynamic forces and moments acting on the UAV's body are easier described in the reference system of the UAV itself.
3) On-board sensors such as an accelerometer and a gyroscope give values relative to the position of the body in space, that is, in the UAV reference system, then the GPS / GLONASS sensors issue values in the Earth's coordinate system.
4) The flight path of the UAV, as well as the set of points to follow, are also specified in the Earth reference system.

Main reference frames used: (fig. 1).

1) Basic inertial frame of reference. Earth reference system. F^i
2) The vehicle frame. F^v. It is achieved by means of a shift relative to F^i.
3) The vehicle-1 frame. UAV reference system with a deviation in the horizontal direction. F^{v1}. It is achieved by turning, with respect to F^v by an angle ψ.
4) The vehicle-2 frame. UAV reference system with a deviation in the horizontal and vertical direction. F^{v2}. Achieved by turning, with respect to F^{v1} by an angle θ.
5) The body frame. UAV reference system with a deviation in the horizontal, vertical direction, and also with the help of rotation relative to the guide axis. F^b. Achieved by turning, relative to F^{v2} by an angle ϕ.
6) The stability frame. The UAV reference system, rotated by the angle of attack. F^s. It is achieved by turning relatively F^b by an angle α.
7) The wind frame. The UAV reference system, rotated by a drift angle by the wind. F^w. It is achieved by turning relatively F^s by an angle β.

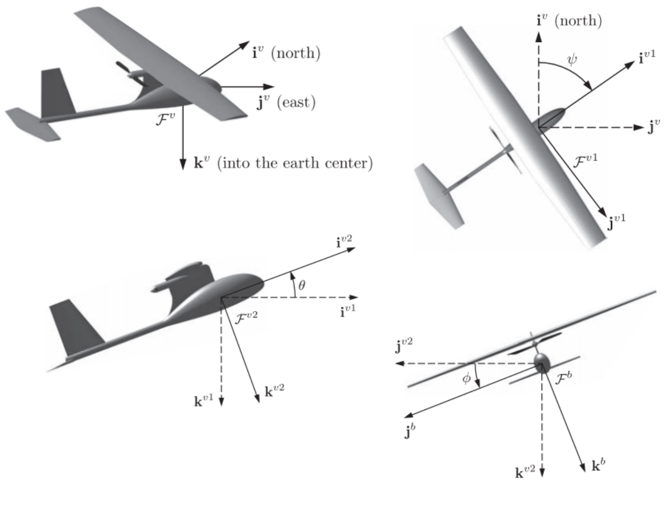


Fig. 1. Different reference frames.

Rotation matrix at angle ν in general form is as follows:

$$\begin{pmatrix} cos(\nu) & sin(\nu) & 0 \\ -sin(\nu) & cos(\nu) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In order to make the transformations between the reference frame of the Earth F^i and the reference system of the UAV F^b, it is necessary to multiply all the intermediate matrices. Thus, the translation matrix will look like this:

$$R_v^b(\phi,\theta,\psi) = R_{v2}^b(\phi)R_{v1}^{v2}(\theta)R_v^{v1}(\psi) =$$

$$\begin{pmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{pmatrix},$$

where $c()$ = $cos()$, $s()$ = $sin()$.

One of the goals of the ready-made autopilot is that the autopilot can bring the UAV to this point with the help of appropriate controls (ailerons, rudders, etc.), that is, it could correctly direct the velocity vector of the UAV [4]. Since the coordinate of the point is indicated in the reference frame of the earth. The first thing to do is to learn how to represent the velocity vector of a UAV in a land reference system. [5]

Suppose the UAV moves in the reference system $\mathcal{F}^b$ relative to the ground (reference frame $\mathcal{F}^i$)) as shown in fig. 2. In this case, the velocity vector **p** is represented as [1]:

$$\mathbf{p} = p_x\mathbf{i}^b + p_y\mathbf{j}^b + p_z\mathbf{k}^b$$

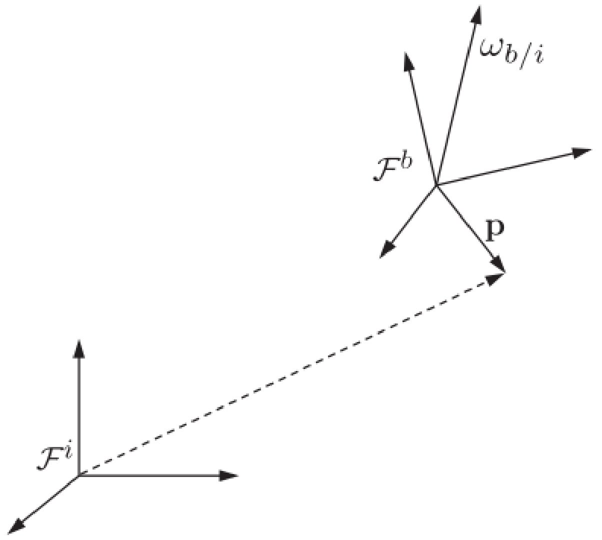


Fig. 2. The vector of forces in different projections.

That is, the motion in time can be given by formula [1]:

$$\frac{d}{dt_b}\mathbf{p} = \dot{p_x}\mathbf{i}^b + \dot{p_y}\mathbf{j}^b + \dot{p_z}\mathbf{k}^b$$

Then the displacement of the vector **p** in the frame of reference $\mathcal{F}^b$ with respect to the reference system $\mathcal{F}^i$ will be given by the formula:

$$\frac{d}{dt_i}\mathbf{p} = \dot{p_x}\mathbf{i}^b + \dot{p_y}\mathbf{j}^b + \dot{p_z}\mathbf{k}^b + p_x\frac{d}{dt_i}\mathbf{i}^b + p_y\frac{d}{dt_i}\mathbf{j}^b + p_z\frac{d}{dt_i}\mathbf{k}^b$$

If we represent the rotation (angular velocity) of $\mathcal{F}^b$ with respect to $\mathcal{F}^i$ as $\omega_{b/i}$, then the increments of the vector in each direction can be represented as:

$$\dot{i}^b = \omega_{b/i} \times \mathbf{i}^b$$

$$\dot{j}^b = \omega_{b/i} \times \mathbf{j}^b$$

$$\dot{k}^b = \omega_{b/i} \times \mathbf{k}^b$$

And if substitute it in the previous equation, it will turn out:

$$p_x{}^b + p_y{}^b + p_z{}^b = p_x(\omega_{b/i}\mathbf{i}^b) + p_y(\omega_{b/i}\mathbf{j}^b) + p_z(\omega_{b/i}\mathbf{k}^b) = \omega_{b/i} \times \mathbf{p}$$

And finally, if we sum up all of the foregoing under one line, then we will get the instant UAV speed expressed through the earth's coordinate system:

$$\frac{d}{dt_i}\mathbf{p} = \frac{d}{dt_b}\mathbf{p} + \omega_{b/i} \times \mathbf{p}$$

B. Kinematics and dynamics

The movement of an unmanned aerial vehicle in a space having 6 degrees of freedom is described using the twelve variables shown in the table:

Name	Description
p_n	The UAV's coordinate axis to north in F^i
p_e	The UAV's coordinate axis to east in F^i
p_d	Axis directed to the center of the Earth in F^i
u	Speed along the axis $i^b inF^b$
v	Speed along the axis $j^b inF^b$
w	Speed along the axis $k^b inF^b$
ϕ	The heeling angle given in F^{v2}
θ	The pitch angle given in F^{v1}
ψ	The yaw angle given in F^v
p	Angular roll speed
q	Angular velocity of pitch
r	Angular speed of yaw

The formulas for the recalculation of these variables for a flying UAV can be found in the following books on mechanics [6], spatial dynamics [7], flight dynamics[8], robotics [9].

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} =$$

$$\begin{pmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - s(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{pmatrix} \times$$

$$\times \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$\begin{pmatrix} \dot{u}_n \\ \dot{v}_e \\ \dot{w}_d \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{m}\begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix}$$

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr \\ \Gamma_5 pr - \Gamma_6(p^2 - r^2) \\ \Gamma_7 pq - \Gamma_1 qr \end{pmatrix} + \begin{pmatrix} \Gamma_3 l + \Gamma_4 n \\ \frac{1}{J_y} m \\ \Gamma_4 l + \Gamma_8 n \end{pmatrix}$$

C. Forces and moments

At the time of flight, the UAV undergoes a non-trivial action of various forces and moments of different natures, namely gravitational (f_g), aerodynamic(f_a, m_a) (f_p, m_p). Then the total action of forces and moments on UAV can be described by formulas [1]:

$$f = f_g + f_a + f_p$$

$$m = m_a + m_p$$

The gravitational force in an inertial coordinate system is described by a simple vector:

$$f_g^v = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix}$$

In the UAV body coordinates system:

$$f_b^v = \begin{pmatrix} -mg * sin(\theta) \\ mg * cos(\theta) sin(\phi) \\ mg * cos(\theta) cos(\phi) \end{pmatrix}$$

When an airplane is flying through the air, it generates its own wings with a lift force and a braking force, as shown in fig. 3. The force and pressure distribution acting on the aircraft depends on the speed through the air, the air density, the shape and position of the aircraft in the air. Thus, the dynamic pressure is described by formula $\frac{1}{2}V_a^2$, where - density of air, and V_a is the airspeed relative to air.

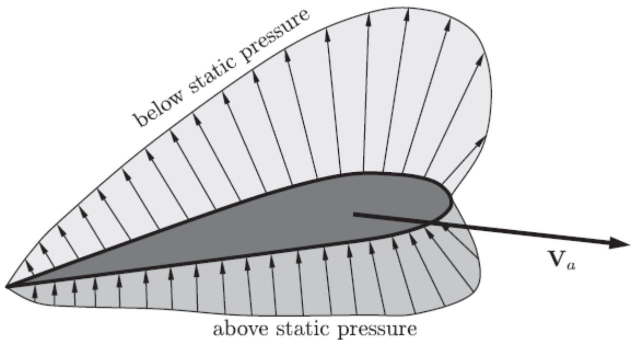


Fig. 3. Distribution of air density around the wing during flight.

The force of inhibition, the force of lifting, and the moment of forces are usually described by the following formulas: [7]

$$F_{lift} = \frac{1}{2}V_a^2 S C_L$$

$$F_{drag} = \frac{1}{2}V_a^2 S C_D$$

$$m = \frac{1}{2}V_a^2 S c C_m,$$

where C_L, C_D, C_m - dimensionless quantities characterizing the aerodynamic coefficients, S - sing surface area, c - value equal to half of the wing.

IV. Implementation

The implementation of the autopilot will be performed on the TRIK platform, shown in fig. 4.

TRIK is a minicomputer compatible with a wide range of peripheral devices, containing all the necessary equipment for creating on its basis autonomous robotic systems. The controller can control the motors of direct current and servo drives, process information from both digital sensors and analog, work with video modules and microphones, has Wi-Fi interfaces, Bluetooth 4.0 (including LE), USB, Micro-SD and ANT. The controller has built-in protection against over-current and from deep battery discharge.[10]

TRIK is based on a processor OMAP-L138 C6-Integra DSP + ARM SoC [11] produced Texas Instruments. The processor has a high-level architecture, shown in fig. 5. This processor has two computational modules:

1) Control ARM core (ARM926EJ-S RISC MPU), which provides the Linux operating system on the controller;
2) core DSP (C674x Fixed/Floating-Point VLIW DSP). It is specially designed to handle a large amount of data represented as vectors.

Fig. 4. Examples of TRIK boards.

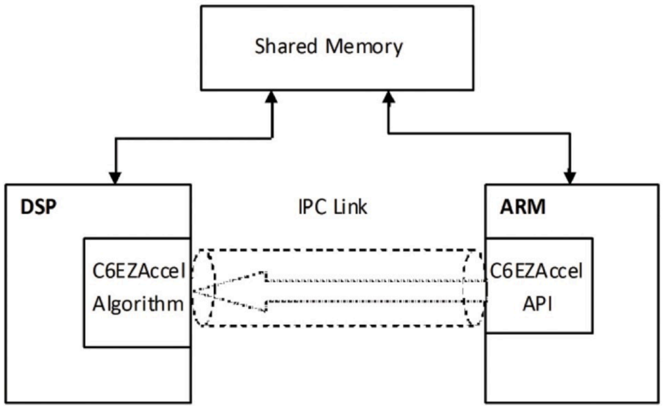


Fig. 5. High-level processor architecture OMAP L138 (ARM+DSP)

V. Control module

The control module operates according to the principle depicted in fig. 6. The Path Planner module specifies the points through which the UAV is scheduled to fly. The Path Manager module converts the sequence of these points into a sequence of lines and arcs (arcs of Dubin), as part of the trajectory over which the UAV is scheduled to fly. Next in the path following, the autopilot itself tracks the passage of the UAV along this path, making adjustments along the route and transferring commands to all means available for route maintenance such as engine, ailerons, rudders, etc.

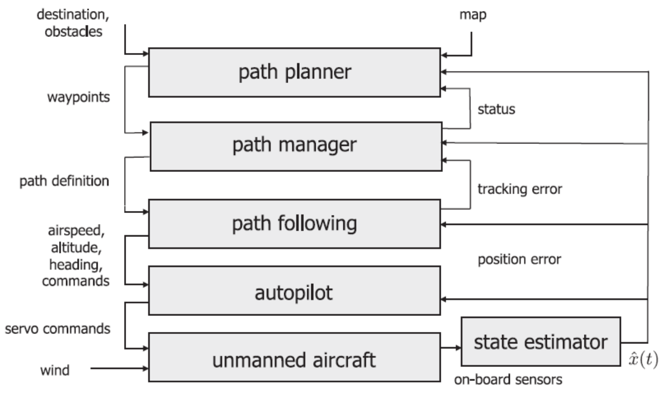


Fig. 6. System architecture of the UAV control module.

There are two classes of problems for planning a path. The first is point-to-point algorithms, the purpose of which is to force the UAV to fly through all given points, bypassing obstacles. The second class of problems are the algorithms for covering a given region using a UAV. For example, for aerial photography. We will focus only on the first class, considering point-to-point algorithms.

Since the control module must take into account the errors and errors that come with the sensor readings, there is a State estimator module for this purpose, which estimates these errors and makes corresponding corrections.

VI. Testing and analysis

Before implementing the proposed architecture on the UAV, the UAV flight simulator was simulated in the Simulink environment of the Matlab application package. Matlab/Simulink is a graphical simulation environment that allows you to build dynamic models, including discrete, continuous and hybrid, non-linear and discontinuous systems, using block diagrams in the form of directed graphs.

The environment is very convenient because it allows you to program differential equations using the built-in S-functions [12]. Draws the position of the UAV in real time, as shown in fig. 7. During the course of the UAV to its Simulink route, it displays all the parameters of all the given physical quantities at any time, which is very convenient, and clearly allows you to check the correctness of the work, as well as analyze the physics of the UAV movement. This is shown in fig. 8. Allows you to specify a complex interaction structure for modules. The main block diagram, which is shown in fig. 9, also contains modules: mavDynamics, responsible for the kinematics and dynamics of the motion process, forcesMoments responsible for the recalculation of forces and moments, drawAircraft - for mapping the UAV in a graphical environment.

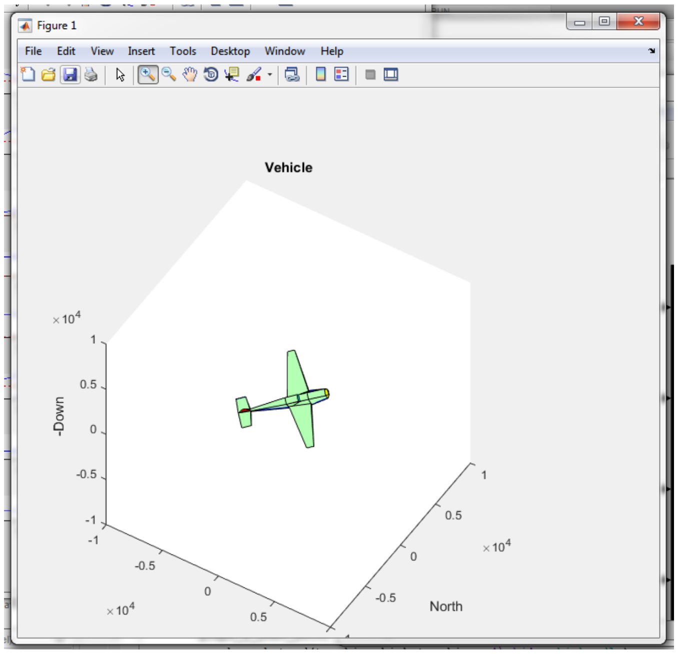


Fig. 7. Displays the UAV model in real time.

VII. Conclusion

The architecture of the autopilot control unit for the UAV was created. It is studied and realized with the help of graphical simulation environment of the physics of the UAV motion process. This includes the basic set of mathematical transformations, which allows you consider the movement of UAVs in different coordinate systems, as well as formulas and accompanying explanations of the dynamics and kinematics of the flight process. The formulas of the forces and the moments operating on the UAV were studied and realized.

The resulting model of motion in the Simulink environment is a good representation of the movement of a real UAV, since it takes into account almost the entire set of forces and moments that affect the UAV, including both constant wind and gusts of wind, presented as white noise.

Further development of the work consists in porting the resulting module to a real UAV, adding an apparatus for estimating the noise of instrument measurements.

Also the received system is a good platform for testing any ideas and hypotheses for UAV, modeling of mathematical processes.

Acknowledgment

This work was supported by RFBR (project 15-08-02640).

References

[1] R. Beard and T. McLain, *Small unmanned aircraft: Theory and practice.*, 2012.

[2] K. Amelin, *The method of orienting an ultralight UAV with a rare update of its location.*

[3] ArduPilot. The site of the.

[4] R. Nelson, *Flight Stability and Automatic Control. Boston, MA: McGrawHill, 2nd ed.*, 1998.

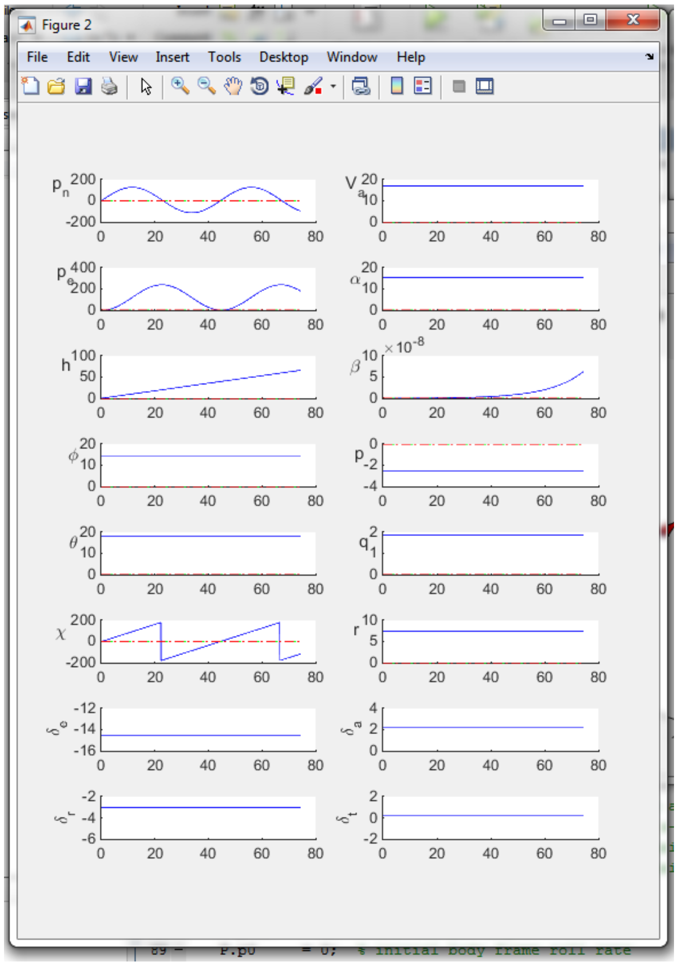


Fig. 8. Matlab/Simulink shows all the specified traffic parameters in real time.

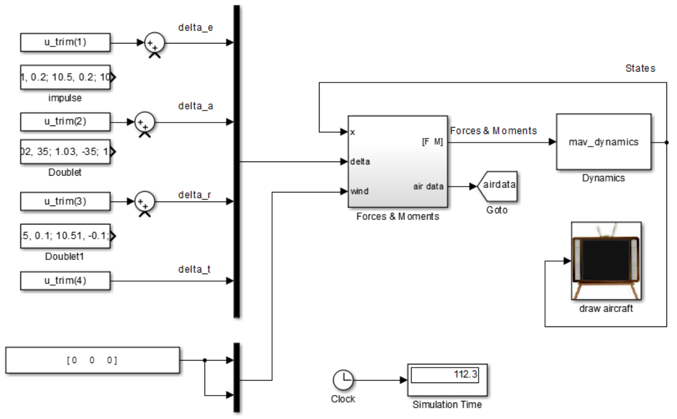


Fig. 9. The main (uppermost) component of the test module, which is a block diagram in the form of directed graphs.

[5] J. Roskam, *Flight Dynamics and Automatic Flight Controls, Parts I II*, 1998.
[6] H. Goldstein, *Classical Mechanics*, 1951.
[7] W. W. E., *Spaceflight Dynamics*, 1997.
[8] M. Shuster, *A survey of attitude representations, The Journal of the Astronautical Sciences, vol. 41, pp. 439517, OctoberDecember*, 1993.
[9] V. M. Spong, M.W., *Robot Dynamics and Control*, 1989.
[10] TRIK. The site of the company.
[11] T. Instruments. Section of the site of the company.
[12] Matlab. (2015) S-function documentation. [Online]. Available: http://www.mathworks.com/help/simulink/matlab-s-functions-1.html