SYRCoSE 2014

Editors:

Alexander Kamkin, Alexander Petrenko and Andrey Terekhov

Preliminary Proceedings of the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering

Saint Petersburg, May 29-31, 2014
Preliminary Proceedings of the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2014), May 29-31, 2014 – Saint Petersburg, Russia:

The issue contains the papers presented at the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2014) held in Saint Petersburg, Russia on May 29-31, 2014. Paper selection was based on a competitive peer review process being done by the program committee. Both regular and research-in-progress papers were considered acceptable for the colloquium.

The topics of the colloquium include formal methods, embedded system design, system programming, process mining, testing, compiler technologies and others.

Предварительный сборник трудов 8-ого весеннего/летнего коллоквиума молодых исследователей в области программной инженерии (SYRCoSE 2014), 29-31 мая 2014 г. – Санкт Петербург, Россия:

Сборник содержит статьи, представленные на 8-ом весеннем/летнем коллоквиуме молодых исследователей в области программной инженерии (SYRCoSE 2014), прошедшем в Санкт Петербурге 29-31 мая 2014 г. Отбор статей производился на основе рецензирования материалов программным комитетом. На коллоквиум допускались как полные статьи, так и краткие сообщения, описывающие текущие исследования.

Программа коллоквиума охватывает следующие темы: формальные методы, проектирование встроенных систем, системное программирование, анализ процессов, тестирование, компиляторные технологии и др.

ISBN 978-5-91474-020-4
Contents

Foreword .............................................................................................................................................. 5
Committees / Referees ......................................................................................................................... 6

Formal Methods

Modular Construction of Time Petri Nets Reachability Graph
I. Knizhnikova, L. Dworzanski ........................................................................................................... 8

On the Deadlock Control in Parallel Resource-Constrained Workflows
V. Bashkin, N. Panfilova ...................................................................................................................... 13

LTL-Specification, Verification and Construction of PLC Programs
D. Ryabukhin, E. Kuzmin ................................................................................................................... 19

An Approach to Lightweight Static Data Race Detection
P. Andrianov, A. Khoroshilov, V. Mutilin ........................................................................................... 27

Minimizing the Number of Static Verifier Traces to Reduce Time for Finding Bugs in Linux Kernel Modules
V. Mordan, E. Novikov ....................................................................................................................... 34

Tools Support for Linux Kernel Deductive Verification Workflow
D. Efremov, N. Komarov .................................................................................................................... 40

Dynamically Allocated Memory Verification in Object-Oriented Programs using Prolog
R. Haberland, S. Ivanovskiy ............................................................................................................... 46

Embedded System Design and System Programming

Energy-Aware Design of Embedded Software through Modelling and Simulation
J.A. Esparza Isasa, P.G. Larsen, F.O. Hansen ..................................................................................... 51

Energy Aware Congestion Management in Dynamic Wireless Mesh Network
S.P. Shiva Prakash, T.N. Nagabhushan, K. Krinkin, O. Sholokhova .................................................... 57

An Architecture of Effective Discrete-Event Simulation Engine for Early Validation of Avionics Systems
D. Buzdalov ........................................................................................................................................ 65

Protecting Applications from Highly Privileged Malware Using Bare-metal Hypervisor
K. Mallachie, N. Pakulin .................................................................................................................... 71

Process Mining

Checking Conformance of High-Level Business Process Models to Event Logs
A. Begicheva, I. Lomazova ................................................................................................................... 77

Applying Graph Grammars for the Generation of Process Models and Their Logs
V. Kataeva, A. Kalenkova ................................................................................................................... 83

Generation of a Set of Event Logs with Noise
I. Shugurov, A. Mitsyuk ...................................................................................................................... 88

DPMine/C: C++ Library and Graphical Frontend for DPMine Workflow Language
S. Shershakov ..................................................................................................................................... 96
Component-based VTMine/C Framework: Not Only Modelling
P. Kim, O. Bulanov, S. Shershakov

Testing

Extended Finite State Machine based Test Derivation Strategies for Telecommunication Protocols
N. Kushik, A. Kolomeez, A. Cavalli, N. Yevtushenko

A Generic Knowledgebase for Test Generation
A. Kotsynyak, A. Tatarnikov

Interactive Test Case Design via Attribute Exploration
F. Strok, G. Kondratiev

Keyword-Driven Testing with Message Sequence Charts
B. Tyutin, A. Veselov, V. Kotlyarov

Reconciliation Testing Aspects of Trading Systems Software Failures
A.-M. Kriger, V. Isayev, A. Pochukalina

Simulation-based Hardware Verification Back-end: Diagnostics
M. Chupilko, A. Protsenko

Compiler Technologies

From Abstract Parsing to Abstract Translation
S. Grigoriev, Ia. Kirilenko

Comparison of Generalized Ascent and Descent Parsers
A. Ragozina, S. Grigoriev

One Approach to Automated Compiler Verification
V. Bessonov, L. Lyadova

Generation of Overlapped Executable Code
V. Aranov, A. Terentiev

Application-Specific Methods and Tools

Predicative Analytics for Developing Software
N. Yarushkina, T. Afanasieva, I. Timina

Detecting and Highlighting Text in Images
I. Pakhomov

Using Multidimensional Ontology of Electronic Document for Solving Semantic Indexing Problem
V. Lanin, G. Sokolov

Generation of Domain-Specific Languages on the Basis of Ontologies
A. Sukhov

Dynamic Information Model Interactions: Design and Implementation of Database-Driven Workflow Approach
A. Petrov
Foreword

Dear participants, we are glad to meet you at the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE). The event is held in Saint Petersburg, the second largest city in Russia and its cultural capital. The colloquium is hosted by Saint Petersburg State Polytechnical University (SPbSPU), one of the top research and educational institutions in Russian Federation in the field of applied physics and mathematics, industrial engineering, chemical engineering, aerospace engineering and many other disciplines. SYRCoSE 2014 is organized by Institute for System Programming of the Russian Academy of Sciences (ISPRAS) and Saint Petersburg State University (SPbSU) jointly with SPbSPU.

In this year, Program Committee (consisting of 50 members from more than 25 organizations) has selected 31 papers. Each submitted paper has been reviewed independently by three referees. Participants of SYRCoSE 2014 represent well-known universities, research institutes and companies such as Aarhus University, Exactpro Systems, ISPRAS, JSS Research Foundation, JSS Academy of Technical Education, Kostroma State Technological University, National Research University – Higher School of Economics, Obninsk Institute for Nuclear Power Engineering, Perm State National Research University, Saint Petersburg Electrotechnical University “LETI”, SPbSPU, SPbSU, TELECOM SudParis, Tomsk State University, Ulyanovsk State Technical University, Yandex, Yaroslavl State University and Yuri Gagarin State Technical University of Saratov (4 countries, 13 cities and 18 organizations).

We would like to thank all of the participants of SYRCoSE 2014 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, Bertrand Meyer (ETH Zürich, Switzerland) and Kostya Serebryany (Google Moscow, Russia). We would also like to thank our sponsors and supporters: Russian Foundation for Basic Research (grant 14-07-06006), Google, Exactpro Systems and CyberLeninka. Finally, our special thanks to local organizers, Igor Chernorutskiy, Vsevolod Kotlyarov and Tatyana Elamic (SPbSPU), for their invaluable help in organizing the colloquium in Saint Petersburg.

Sincerely yours

Alexander Kamkin, Alexander Petrenko and Andrey Terekhov
May 2014
# Committees

## Program Committee Chairs

<table>
<thead>
<tr>
<th>Chairs</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexander Petrenko – Russia</td>
<td>Institute for System Programming of RAS</td>
</tr>
<tr>
<td>Andrey Terekhov – Russia</td>
<td>Saint-Petersburg State University</td>
</tr>
</tbody>
</table>

## Program Committee

<table>
<thead>
<tr>
<th>Members</th>
<th>Institutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jean-Michel Adam – France</td>
<td>Pierre Mendès France University</td>
</tr>
<tr>
<td>Sergey Avdoshin – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Eduard Babkin – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Svetlana Chuprina – Russia</td>
<td>Perm State National Research University</td>
</tr>
<tr>
<td>Pavel Drobitshev – Russia</td>
<td>Saint-Petersburg State Polytechnic University</td>
</tr>
<tr>
<td>Liliya Emaletdinova – Russia</td>
<td>Institute for Technical Cybernetics and Informatics, KNRTU</td>
</tr>
<tr>
<td>Victor Gergel – Russia</td>
<td>Lobachevsky State University of Nizhny Novgorod</td>
</tr>
<tr>
<td>Efim Grinkrug – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Maxim Gromov – Russia</td>
<td>Tomsk State University</td>
</tr>
<tr>
<td>Vladimir Hahanov – Ukraine</td>
<td>Kharkov National University of Radioelectronics</td>
</tr>
<tr>
<td>Shihong Huang – USA</td>
<td>Florida Atlantic University</td>
</tr>
<tr>
<td>Iosif Itkin – Russia</td>
<td>Exactpro Systems</td>
</tr>
<tr>
<td>Alexander Kamkin – Russia</td>
<td>Institute for System Programming of RAS</td>
</tr>
<tr>
<td>Vsevolod Kotlyarov – Russia</td>
<td>Lobachevsky State Polytechnic University</td>
</tr>
<tr>
<td>Oleg Kozyrev – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Vladimir Kozyrev – Russia</td>
<td>National Research Nuclear University “MEPhI”</td>
</tr>
<tr>
<td>Daniel Kurushin – Russia</td>
<td>State National Research Polytechnic University of Perm</td>
</tr>
<tr>
<td>Peter Gorm Larsen – Denmark</td>
<td>Aarhus University</td>
</tr>
<tr>
<td>Rustam Latypov – Russia</td>
<td>Institute of Computer Science and Information Technologies, KFU</td>
</tr>
<tr>
<td>Alexander Letichevsky – Ukraine</td>
<td>Glushkov Institute of Cybernetics, NAS</td>
</tr>
<tr>
<td>Alexander Lipanov – Ukraine</td>
<td>Kharkov National University of Radioelectronics</td>
</tr>
<tr>
<td>Irina Lomazova – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Ludmila Lyadova – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Victor Malysenko – Russia</td>
<td>Moscow State University</td>
</tr>
<tr>
<td>Vladimir Makarov – Russia</td>
<td>Yaroslavl-the-Wise Novgorod State University</td>
</tr>
<tr>
<td>Tiziana Margaria – Germany</td>
<td>University of Potsdam</td>
</tr>
<tr>
<td>Marek Milosz – Poland</td>
<td>Institute of Computer Science, Lublin University of Technology</td>
</tr>
<tr>
<td>Alexey Namestnikov – Russia</td>
<td>Ulyanovsk State Technical University</td>
</tr>
<tr>
<td>Valery Nepomniashchy – Russia</td>
<td>Ershov Institute of Informatics Systems</td>
</tr>
<tr>
<td>Mykola Nikitchenko – Ukraine</td>
<td>Kyiv National Taras Shevchenko University</td>
</tr>
<tr>
<td>Yuri Okulovsky – Russia</td>
<td>Ural Federal University</td>
</tr>
<tr>
<td>Elena Pavlova – Russia</td>
<td>Tomsk State University</td>
</tr>
<tr>
<td>Victor Malyshko – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Ivan Pilotse – Belorusia</td>
<td>Belarusian State University of Informatics and Radioelectronics</td>
</tr>
<tr>
<td>Vladimir Popov – Russia</td>
<td>Ural Federal University</td>
</tr>
<tr>
<td>Yury Rogozov – Russia</td>
<td>Taganrog Institute of Technology, Southern Federal University</td>
</tr>
<tr>
<td>Rustam Sabitov – Russia</td>
<td>Kazan National Research Technical University</td>
</tr>
<tr>
<td>Nikolay Shilov – Russia</td>
<td>Ershov Institute of Informatics Systems</td>
</tr>
<tr>
<td>Ruslan Smelyansky – Russia</td>
<td>Moscow State University</td>
</tr>
<tr>
<td>Valeriy Sokolov – Russia</td>
<td>Yaroslavl State Technical University</td>
</tr>
<tr>
<td>Petr Sosnin – Russia</td>
<td>Ulyanovsk State Technical University</td>
</tr>
<tr>
<td>Veniamin Tarasov – Russia</td>
<td>Povolzhsky State University of Telecommunications and Informatics</td>
</tr>
<tr>
<td>Sergey Ustinov – Russia</td>
<td>Saint-Petersburg State Polytechnic University</td>
</tr>
<tr>
<td>Vladimir Voevodin – Russia</td>
<td>Research Computing Center of Moscow State University</td>
</tr>
<tr>
<td>Dmitry Volkov – Russia</td>
<td>Moscow State University</td>
</tr>
<tr>
<td>Mikhail Volkov – Russia</td>
<td>Ural Federal University</td>
</tr>
<tr>
<td>Nadezhda Yavorsky – Russia</td>
<td>Ulyanovsk State Technical University</td>
</tr>
<tr>
<td>Rostislav Yavorsky – Russia</td>
<td>National Research University Higher School of Economics</td>
</tr>
<tr>
<td>Nina Yevtushenko – Russia</td>
<td>Tomsk State University</td>
</tr>
<tr>
<td>Vladimír Zakharov – Russia</td>
<td>Moscow State University</td>
</tr>
<tr>
<td>Sergey Zaitsev – Russia</td>
<td>Kazan National Research Technical University</td>
</tr>
</tbody>
</table>

## Organizing Committee Chairs and Secretaries

<table>
<thead>
<tr>
<th>Chairs</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Igor Chernorutkiy – Russia</td>
<td>Saint-Petersburg State Polytechnic University</td>
</tr>
<tr>
<td>Tatiana Elamiy – Russia</td>
<td>Saint-Petersburg State Polytechnic University</td>
</tr>
<tr>
<td>Alexander Kamkin – Russia</td>
<td>Institute for System Programming of RAS</td>
</tr>
<tr>
<td>Alexander Petrenko – Russia</td>
<td>Institute for System Programming of RAS</td>
</tr>
<tr>
<td>Vsevolod Kotlyarov – Russia</td>
<td>Saint-Petersburg State Polytechnic University</td>
</tr>
</tbody>
</table>
# Referees

<table>
<thead>
<tr>
<th>Jean-Michel Adam</th>
<th>Tiziana Margaria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sergey Avdoshin</td>
<td>Ivan Mikhailov</td>
</tr>
<tr>
<td>Eduard Babkin</td>
<td>Alexey Namestnikov</td>
</tr>
<tr>
<td>Mikhail Chupilko</td>
<td>Mykola Nikitchenko</td>
</tr>
<tr>
<td>Svetlana Chuprina</td>
<td>Yuri Okulovsky</td>
</tr>
<tr>
<td>Pavel Drobintsev</td>
<td>Nikolay Pakulin</td>
</tr>
<tr>
<td>Denis Efremov</td>
<td>Elena Pavlova</td>
</tr>
<tr>
<td>Victor Gergel</td>
<td>Alexander Petrenko</td>
</tr>
<tr>
<td>Efim Grinkrug</td>
<td>Ivan Piletski</td>
</tr>
<tr>
<td>Maxim Gromov</td>
<td>Vladimir Popov</td>
</tr>
<tr>
<td>Shihong Huang</td>
<td>Yury Rogozov</td>
</tr>
<tr>
<td>Iosif Itkin</td>
<td>Nikolay Shilov</td>
</tr>
<tr>
<td>Dmitry Ivankov</td>
<td>Sergey Smolov</td>
</tr>
<tr>
<td>Anna Kalenkova</td>
<td>Valeriy Sokolov</td>
</tr>
<tr>
<td>Alexander Kamkin</td>
<td>Petr Sosnin</td>
</tr>
<tr>
<td>Dmitry Kosolobov</td>
<td>Veniamin Tarasov</td>
</tr>
<tr>
<td>Vsevolod Kotlyarov</td>
<td>Andrei Tatarnikov</td>
</tr>
<tr>
<td>Artem Kotsynyak</td>
<td>Andrey Terekhov</td>
</tr>
<tr>
<td>Vladimir Kozyrev</td>
<td>Dmitry Volkov</td>
</tr>
<tr>
<td>Daniel Kurushin</td>
<td>Mikhail Volkov</td>
</tr>
<tr>
<td>Peter Gorm Larsen</td>
<td>Nadezhda Yarushkina</td>
</tr>
<tr>
<td>Alexander Lipanov</td>
<td>Rostislav Yavorskiy</td>
</tr>
<tr>
<td>Irina Lomazova</td>
<td>Nina Yevtushenko</td>
</tr>
<tr>
<td>Lyudmila Lyadova</td>
<td>Vladimir Zakharov</td>
</tr>
<tr>
<td>Victor Malyshko</td>
<td></td>
</tr>
</tbody>
</table>
Checking Conformance of High-Level Business Process Models to Event Logs

Antonina K. Begicheva
National Research University
Higher School of Economics (HSE)
33 Kirpichnaya Str., Moscow, Russia
Email: be-ton@yandex.ru

Irina A. Lomazova
National Research University
Higher School of Economics (HSE)
33 Kirpichnaya Str., Moscow, Russia
Email: ilomazova@hse.ru

I. INTRODUCTION

Process mining \cite{1} is a new technology, providing a variety of methods to discover, monitor and improve real processes by extracting knowledge from event logs. The two most prominent process mining tasks are: (i) process discovery: constructing a process model from example behavior recorded in an event log, and (ii) conformance checking: diagnosing and quantifying discrepancies between observed behavior and modeled behavior. There are many software products which allow us to use methods of Process Mining. ProM \cite{4} is an open-source tool supporting many techniques of Process Mining, which are represented as plug-ins. Due to a flexibility of this environment it can be used both for research and applications.

This paper studies conformance checking \cite{1, 3, 6, 7}. Conformance checking uses both an event log and a model, and compares observed behavior written in the log with the behavior produced by the model. The general goal is to find discrepancies between them to improve a model. Conformance checking techniques can also be used for measuring the performance of process discovery algorithms (that restores a model on the basis of a known log) and to repair models that have no a well alignment with the real behavior of the process.

There are four model’s evaluation criteria: fitness, precision, generalization and simplicity. Fitness measures “the proportion of behavior in the event log possible according to the model”. Among the four quality criteria, fitness is the most related to the conformance. Several methods of conformance checking were developed. We consider methods based on replay approach \cite{3}. Replaying a log on a model can help to measure fitness.

An obvious approach to measure fitness would be just to count the fraction of cases that can be "parsed completely" (i.e. the proportion of cases corresponding to firing sequences leading from \texttt{[start]} to \texttt{[end]}). Fitness can range from 0 to 1. It is supposed that fitness is equal to 1, if the log perfectly fits the model. When measuring fitness by replaying, we could stop replaying a trace when we face a problem and mark this trace as unsuitable. We get more information about conformance if we continue replaying the trace on the model, and record a count of all missing tokens and all tokens that are pending at the end.

Let us denote the number of produced tokens by \( p \), the number of consumed tokens by \( c \), the number of missing tokens by \( m \) and the number of remaining tokens by \( r \). Initially, when all places are empty, \( p = c = 0 \). Then the environment produces a token for the place \texttt{[start]}, Therefore, the \( p \) counter is incremented: \( p \leftarrow p + 1 \). A log is replayed by consecutive firings of transitions, corresponding to activities of the process. Each transition consumes and produces several tokens and we increase the corresponding variables. If we need an extra token in a place to continue replaying (when the next transition is not enabled), then the \( m \) counter must be incremented and the place, that lacks a token, is marked as a place where a token was missed. If by the time of consuming a token from the place \texttt{[end]} there were tokens pending in some other places, the \( r \) counter must be increased by the number of the remained tokens, and the places with tokens must be tagged.

The fitness of a trace \( \sigma \) of a workflow process model \( N \) is defined as follows:

\[
fitness(\sigma, N) = \frac{1}{2} (1 - \frac{m}{p}) + \frac{1}{2} (1 - \frac{r}{p})
\]

When working with business processes we typically use detailed logs, which present the full report about sequentially executed activities. Since in most information systems logs are generated automatically, keeping detailed records is not a problem. However, large and detailed models are not good to deal with. Such models are not clear and readable for experts. Experts prefer to work with more abstract (high-level) models. More abstract models are easier to construct, understand and analyze. Process models developed by people are, as a rule, not very large and abstract from technical details. So, checking conformance of an abstract model and a low-level event log, generated by an information system, is an important and challenging problem. However, as far as we know, this problem was not studied in the literature.

In this paper we consider an abstract model in which each separate activity represents a subprocess built from a set of smaller activities. A history of a detailed process behavior is recorded in low-level logs. Process models are represented by workflow nets — a special subclass of Petri nets \cite{2}. We present a method for checking conformance of an abstract model and a low-level event log.

The paper is organized as follows. Section II contains some basic definitions and notions, including Petri nets, event log, perfect fits and refinement. In Section III we give a motivating example of handling a request for a compensation within airline in terms of Petri nets. In Section IV we present a method for checking conformance between an abstract model and a low-level log. We also give a justification of this method.
by proving its correctness in the case of perfect fitness. An implementation of our algorithm is described in Section V. Section VI contains some conclusions.

II. PRELIMINARIES

We start with recalling some basic notions from the set theory. Let \( S \) be a set. By \( S^* \) we denote the set of all finite sequences (words) over \( S \).

\[
S = S_1 \cup S_2 \cup \ldots \cup S_n \quad \text{is a partition of} \quad S \quad \text{iff} \quad \forall i, j \in [1, n]: S_i \subseteq S \quad \text{and} \quad S_i \cap S_j = \emptyset.
\]

A multiset \( m \) over a set \( S \) is a mapping \( m : S \to \mathbb{N} \), where \( \mathbb{N} \) is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

For two multisets \( m, m' \) we write \( m \leq m' \) iff \( \forall s \in S : m(s) \leq m'(s) \) (the inclusion relation). The sum of two multisets \( m \) and \( m' \) is defined as usual: \( \forall s \in S : (m + m')(s) = m(s) + m'(s) \), the difference is a partial function: \( \forall s \in S \) such that \( m(s) \geq m'(s) : (m - m')(s) = m(s) - m'(s) \). By \( \mathcal{M}(S) \) we denote the set of all finite multisets over \( S \). Non-negative integer vectors are often used to encode finite multisets.

**Definition 1** (Petri net). Let \( P \) and \( T \) be disjoint finite sets of places and transitions and \( F : (P \times T) \cup (T \times P) \to \mathbb{N} \). Then \( N = (P, T, F) \) is a Petri net. Let \( A \) be a finite set of activities. A labeled Petri net is a Petri net with a labeling function \( \lambda : T \to A \cup \{\epsilon\} \) which maps every transition to an activity (a transition label) from \( A \), or a special label \( \epsilon \), corresponding to an invisible action.

A marking in a Petri net is a function \( m : P \to \mathbb{N} \), mapping each place to some natural number (possibly zero). Thus a marking may be considered as a multiset over the set of places. Pictorially, \( P \)-elements are represented by circles, \( T \)-elements by boxes, and the flow relation \( F \) by directed arcs. Places may carry tokens represented by filled circles. A current marking \( m \) is designated by putting \( m(p) \) tokens into each place \( p \in P \).

For a transition \( t \in T \) an arc \( (x, t) \) is called an input arc, and an arc \( (t, x) \) — an output arc; the preset \( t^* \) and the postset \( t^+ \) are defined as the multisets over \( P \) such that \( t^*(p) = F(p, t) \) and \( t^+(p) = F(t, p) \) for each \( p \in P \).

A transition \( t \in T \) is enabled in a marking \( m \) iff \( \forall p \in P : m(p) \geq F(p, t) \). An enabled transition \( t \) may fire yielding a new marking \( m' = m - t^* + t^+ \), i.e. \( m'(p) = m(p) - F(p, t) + F(t, p) \) for each \( p \in P \) (denoted \( m \xrightarrow{t} m' \), \( m \xrightarrow{\lambda(t)} m' \), or just \( m \to m' \)).

A Workflow-net is a (labeled) Petri net with two special places: \( i \) and \( f \). These places are used to mark the beginning and the ending of a workflow process.

**Definition 2** (Workflow net). A (labeled) Petri net \( N = (P, T, F, \lambda) \) is called a workflow net (WF-net) iff

1. There is one source place \( i \in P \) and one sink place \( f \in P \) s.t. \( i^* = f^* = \emptyset \);
2. Every node from \( P \cup T \) is on a path from \( i \) to \( f \);
3. The initial marking in \( N \) contains the only token in its source place.

![Fig. 1. Extending a WF net with initial and final transitions](image)

By abuse of notation we denote by \( i \) both the source place and the initial marking in a WF-net. Similarly, we use \( f \) to denote the final marking in a WF-net \( N \), defined as a marking containing the only token in the sink place \( f \).

Let \( N = (P, T, F, \lambda) \) be a WF-net. The extended WF net (EWF-net) \( N' = (P', T', F', \lambda') \) is defined as follows: \( P' = P, T' = T \cup \{t_i, t_f\} \), and \( F' = F \cup \{\langle t_i, i \rangle, \langle f, t_f \rangle\} \), where \( t_i, t_f \) are new (not occurring in \( P, T \)) nodes. The new transitions \( t_i, t_f \) are labeled with invisible activity \( \epsilon \) in \( N' \), all other transitions in \( N' \) have the same labels as in \( N \). In the remainder we will denote such an extended WF net of \( N \) as \( t^+(N) \). The initial marking in an extended WF net contains no tokens. Thus an extended WF net may start a new case at any moment (cf. Fig. 1).

Event logs keep a history of process executions.

**Definition 3** (Event log). Let \( A \) be a finite set of activities. A trace \( \sigma \) is a finite sequence of activities, i.e., \( \sigma \in A^* \). An event log \( L \) is a finite multiset of traces, i.e., \( L \in \mathcal{M}(A^*) \).

In this paper we study conformance checking. Given a model and an event log we would like to compare the process model behavior and the behavior recorded in the event log. Several metrics for conformance checking were defined in the literature [1]. Among the most important metrics is fitness. Informally speaking, fitness measures the proportion of behavior in the event log possible according to the model.

**Definition 4** (Perfect fit). Let \( N \) be a WF-net with transition labels from \( A \), an initial marking \( i \), and a final marking \( f \). Let \( \sigma \) be a trace over \( A \). We say that a trace \( \sigma = a_1, \ldots, a_k \) perfectly fits \( N \) iff there exists a sequence of firings \( i \xrightarrow{a_1} \ldots \xrightarrow{a_k} f = N, s.t. the sequence of activities \( \lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k) \) after deleting all invisible activities \( \epsilon \) coincides with \( \sigma \). A log \( L \) perfectly fits \( N \) iff every trace from \( L \) perfectly fits \( N \).

Petri nets can be extended with hierarchy and it is done e.g. in Colored Petri nets (CPN) [8]. Hierarchy allows to develop more compact models with a compositional network structure. In the case of two-level hierarchy there are two models of one process: a high-level (abstract) model and a low-level (refined) model. The high-level model is a model with abstract transitions. An abstract transition refers to a Petri net subprocess model refining the activity represented by this transition. The low-level model can be obtained from an abstract model by substituting subprocess models for abstract transitions.

**Definition 5** (Substitution). Let \( N_1 = (P_1, T_1, F_1, \lambda_1) \) be a WF-net, \( t \in T \) be a transition in \( N_1 \). Let also \( N_2 = \)
(P2, T2, F2, λ2) be an EWF-net with the initial and final transitions t1, t7 correspondingly. We say that a WF-net N3 = (P3, T3, F3, λ) is obtained by a substitution [t → N2] of N2 for t in N1 iff P3 = P1 ∪ P2, T3 = T1 ∪ T2 \ {t}, F3 = F1 ∪ F2 \ {{(p, t) | p ∈ \{t\}} \cup \{(p, t1) | p ∈ \{t1\}} \cup \{(t, p) | p ∈ \{t\}}.}

**Definition 6** (Refinement). Let Nₐ, Nᵣ be two WF-nets with sets of activities Aₐ, Aᵣ correspondingly. Let Aᵣ = a₁, a₂, ..., aₙ, and Aₐ = A₁ᵣ ∪ A₂ᵣ ∪ ... ∪ Aₙᵣ be a partition of Aᵣ into \( n \) subsets, and N₁, N₂, ..., Nₙ be EWF-nets with sets of activities A₁ᵣ, ..., Aₙᵣ correspondingly. We say that Nᵣ is a refinement of Nₐ via substitutions \( \{a_1 → N₁ᵣ, a_2 → N₂ᵣ, ..., aₙ → Nₙᵣ\} \) iff Nᵣ can be obtained from Nₐ by simultaneous substitutions of Nᵣ for all t s.t. λ(t) = a₁.

### III. Motivating Example

Let us consider a toy model from \[1\], which describes handling a request for a compensation within airline. Here customers may request compensations for various reasons. An abstract model of this process (expressed in terms of a Petri net) is presented in Fig.2. Fig.3 presents a refined model of the same process. To avoid congestion of activities' names in the low-level model in Fig.3 only places inherited from the abstract model are labeled in the picture.

![Fig. 2. An abstract model for handling compensation requests](image)

![Fig. 3. A refined model for handling compensation requests, which refines the model in Fig. 2](image)

Let us explain the correspondence between the two models. Let N₀, N¹, N², N³, N⁴, N⁵, N⁶, N⁷ be EWF-nets with sets of activities A⁰ = \{t0, t1, t2, t3\}, A¹ = \{t4, t5, t6, t7, t8, t9\}, A² = \{t10, t11, t12\}, A³ = \{t13, t14, t15\}, A⁴ = \{t16, t17, t18\}, A⁵ = \{t19, t20, t21\}, A⁶ = \{t22, t23, t24\}, A⁷ = \{t25, t26, t27\} correspondingly. The refined model in Fig.3 is the refinement of the abstract model in Fig.2 via the substitutions request → N₀, examine_thoroughly → N₁, examine_casually → N₂, check_ticket → N₃, decide → N₄, reinitiate_request → N₅, pay_compensation → N₆, reject_request → N₇. A sample of an event log obtained for the refined model Lᵣ is shown in Fig.4. Note that for the log Lᵣ (Fig.4) and the refined model (Fig.3) we have fitness = 1, since the log Lᵣ is generated by the model.

So, we have an abstract model (as a more simple to understand and analyse) and we want to check conformance of this model to a low-level log. It is obvious that we cannot do it straightforward, since the model is defined in terms of abstract activities, and the log contains low-level activities.

### IV. Checking Conformance between an Abstract Model and a Refined Event Log

To check conformance between an abstract model and a low-level log, we first transform the given log into a log over abstract activities. For this purpose, each low-level activity in the log is replaced by a name of the subprocess (an abstract
activity) it belongs to. Hence we get a log with "stuttering" abstract activities. This transformation is implemented by the method `toHighLevel()`, schematically presented in Algorithm 1.

**Data:** lowlevellog — a list of low-level activities, $haction$ — a set of high-level activities, where for each high-level activity is stored information about its partition into subsets of low-level activities.

**Result:** highlevellog — a high-level event log.

```plaintext
i ← 0;
highlevellog ← ∅;
currentLowAction ← lowlevellog[i] while i < lowlevellog.size do
    // search of high-level activity, subsets of which contains this low-level activity
    currentHighAction ← search(haction, currentLowAction);
    if currentHighAction ≠ ∅ then
        // check of condition that low-level activity is included to partition of the current high-level action
        while i < lowlevellog.size and currentHighAction.contains(currentLowAction) do
            i ← i + 1;
        end
        highlevellog.add(currentHighAction);
    end
end
return highlevellog;
```

**Algorithm 1:** Method `toHighLevel()`, transforming a low-level log into a log over abstract activities

---

After converting the refined log into notations of the abstract model we get a new log, which is a multiset of sequences of abstract activities. But this still cannot be used for the conformance checking because of stuttering actions. Moreover, when we have two concurrent subprocesses, represented by two concurrent abstract activities in an abstract model, stuttering sequences may interleave. To overcome this problem we transform an abstract model into a model allowing stuttering of each abstract activity. For this purpose we add loops to transitions in the abstract model.

Algorithm 2 schematically describes the method `addLoops()` for transforming an abstract model by adding loops to abstract transitions (cf. Fig. 5).

Now we describe the general algorithm of conformance checking between an abstract model and a low-level log in more details.

**Main Algorithm** (Converting a refined log to an abstract one and transforming the high-level model for working with result of this conversion).

Let $N_a = (P,T,F,λ)$ be a Petri-net corresponding to an abstract model of a process over a set of activities $A_a$. Let also $L_r$ be a low-level event log (a finite multiset of traces) over a set $A_r$ of low-level activities.

Let $A_a, A_r$ be the set of activities in the abstract model and a set of activities in a refined model correspondingly. Denote by and denote trace from it by $σ_j^r ∈ L_r$ where $j$ is a number of trace.

1. Convert $L_r$ to a high-level event log (denote it $L_a$) using data about the partition of $A_r$. Replace each $a_k ∈ σ_j^r$ (where $k$ is a number of activities in the trace) to the corresponding activity from $A_a$ for all $j$ and $k$.
2. Use a rule introduced by us about repetitive activities in $L_a$. Replace every sequences of identical activities
Algorithm 2 to the abstract model $N_a$ is denoted by applying Algorithm 1. The log obtained as the result of this by replaying traces from $L_a$ in $N_a$. It turns out, that all traces from $L_a$ can be replayed in $N_a$, i.e. the log $L_a$ perfectly fits $N_a$. This is not by chance. The following theorem states, that the proposed conformance checking method is stable under perfect fitness. 

**Theorem 1.** Let $N_a$ be a refinement of $N_a$ and $L_r$ be an event log over the set of activities $A_r$, i.e. $L_r \in \mathcal{M}(A_r^+).$ If $L_r$ perfectly fits $N_r$, then the main algorithm return 1, which is interpreted as $L_r$ perfectly fits $N_a$.

We omit the proof of the theorem, since it is rather technical and straightforward.

V. IMPLEMENTATION

The proposed method for checking conformance of high-level business model to low-level event log is implemented as a plug-in for ProM.

Our tool consists of six main classes:

1) `TransformerForConformanceChecking` class is responsible for interaction with framework and GUI.
2) `HighLevelTransition` class represents a high-level transition. Each object of this type have a name of appropriate abstract activity and an array of low-level activities, corresponding to this object.
3) `Activity` class represents an activity and implements forming of activity with data from event log.
4) `ConverterForLowLevelLog` class is responsible for implementation of Algorithm 1, i.e. it transforms a low-level event log to an abstract event log.
5) `ConverterForModel` class is responsible for implementation of Algorithm 2, i.e. it transforms an abstract model by adding the requisite loops.

VI. CONCLUSION

Abstract models are much more clear and more easily understood than low-level models. But in practice we have only low-level logs, which cannot be used for direct conformance checking. Hence checking conformance of a high-level business model to a low-level event log is an important task to facilitate the expert’s work. In this paper we have presented a method for solving this problem. Also we had developed a ProM plug-in which implements the proposed algorithm.

We have proved, that our method recognizes perfect fitness between an abstract model and a low-level log correctly. This can be considered as a justification of the proposed approach. However, this is not enough. It is very important to check the method on logs with deviations. In the further research we plan experiments with different logs (logs with noise and different kinds of deviations), as well as real application logs, and we shall work on improving the algorithms through the use of found heuristics.

ACKNOWLEDGMENT

This study was carried out within the National Research University Higher School of Economics’ Academic Fund.
\[ L = \{ <\text{register request, examine thoroughly, check ticket, examine thoroughly, check ticket, examine thoroughly, decide, reject request}>, \\
<\text{register request, examine thoroughly, check ticket, examine thoroughly, check ticket, examine thoroughly, decide, pay compensation}>, \\
<\text{register request, check ticket, examine casually, check ticket, examine casually, decide, pay compensation}>, \\
<\text{register request, check ticket, examine thoroughly, check ticket, examine thoroughly, decide, reject request}>, \\
<\text{register request, examine thoroughly, check ticket, examine thoroughly, decide, pay compensation}>, \\
<\text{register request, examine casually, check ticket, examine casually, decide, pay compensation}>, \\
<\text{register request, check ticket, examine casually, check ticket, examine casually, decide, reject request}>, \\
<\text{register request, check ticket, examine casually, check ticket, decide, pay compensation}>, \\
<\text{register request, check ticket, examine casually, check ticket, decide, pay compensation}>, \\
<\text{register request, check ticket, examine casually, check ticket, examine casually, decide, reject request}>, \} \]

Fig. 6. The abstract event log obtained by applying Algorithm 1 to the initial event log in Fig. 4

Fig. 7. The abstract model after adding loops (by applying Algorithm 2 to the model in Fig. 2)

REFERENCES