Scientific Papers of the Polish Information Processing Society Scientific Council

Software Engineering Research for the Practice

Scientific Editors:
Piotr Kosiuczenko, Lech Madeyski, Mirosław Ochodek, Andrzej Paszkiewicz

SCIENTIFIC COUNCIL
POLISH INFORMATION PROCESSING SOCIETY

PTI

Software Engineering Research for the Practice

Scientific Editors

Piotr Kosiuczenko, Lech Madeyski,

Mirosław Ochodek, Andrzej Paszkiewicz

Authors

Anna Derezińska – ***CHAPTER 1***
Wiktor Nowakowski, Kamil Rybiński, Michał Śmiałek – ***CHAPTER 2***
Andrzej Zalewski, Andrzej Ratkowski, Małgorzata Purwin – ***CHAPTER 3***
Marcin Daszuta, Dominik Szajerman, Piotr Napieralski – ***CHAPTER 4***
Michał Wroński – ***CHAPTER 5***
Grzegorz Kochański – ***CHAPTER 6***
Aneta Poniszewska-Marańda, Joanna Pawelska, Aneta Majchrzycka – ***CHAPTER 7***
Radosław Markiewicz, Ziemowit Nowak – ***CHAPTER 8***
Andrzej Turnau, Wojciech Zwonarz – ***CHAPTER 9***
Wiktor Daszczuk, Maciej Bielecki, Jan Michalski – ***CHAPTER 10***
Marek Bolanowski, Andrzej Paszkiewicz – ***CHAPTER 11***
Mateusz Kut, Bartosz Pawłowicz, Bartosz Trybus – ***CHAPTER 12***

Reviewers

Zbigniew Banaszak, Marek Bolanowski, Rémy Dupas, Mel Ó Cinnéide, Krzysztof Goczyła, Janusz Górski, Szymon Grabowski, Bogumiła Hnatkowska, Zbigniew Huzar, Stanisław Jarząbek, Piotr Kosiuczenko, Natalia Kryvinska, Tadeusz Kwater, Wojciech Kwedlo, Kevin Lano, Tomasz Lewowski, Leszek Maciaszek, Lech Madeyski, Piotr Napieralski, Mirosław Ochodek, Aneta Poniszewska-Marańda, Łukasz Radliński, Adam Roman, Małgorzata Sadowska, Michał Śmiałek, Zenon Sosnowski, Mirosław Staron, Andrzej Stasiak, Jakub Swacha, Tomasz Szmuc, Leszek Trybus, Bartosz Walter, Andrzej Wąsowski, Andrzej Zalewski, Tomasz Żabiński.

Scientific Editors

Piotr Kosiuczenko, Lech Madeyski, Mirosław Ochodek, Andrzej Paszkiewicz

ISBN: 978-83-946253-6-8

Edition: I. Copies: 100. Publishing sheets: 10,6. Print sheets: 11.
Print: Oficyna Wydawnicza Politechniki Rzeszowskiej
al. Powstańców Warszawy 12, 35-959 Rzeszów

Contents

Preface

Software systems are playing more and more important role each year. According to Gartner Consulting, the global IT spending is expected to exceed $3.5 trillion (i.e., 3.5 * 10^12) by the end of 2017. It is also expected that the software market is going to grow 7.2% in 2017 to reach the total of $357 billion (i.e., 357 * 10^9).

Such rapid growth in demand for software-intensive products forces software development companies to release their product faster and under pressure. This makes the software development a cumbersome task. Unfortunately, even a small mistake made by software developers could result in expensive failures and disasters such as the crashes of space rockets Mariner 1 or Ariane 5. In extreme cases, a software failure can even cost human lives, like in the case of Therac-25 or Patriot Missile Failures during the Gulf War.

The goal of Software Engineering is to support software developers in delivering high-quality products on-time and within the budgets. Although it is a relatively young engineering discipline, many of the proposed methods have become de-facto standards. The IEEE Computer Society defines Software Engineering as: "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software." Thus, Software Engineering intertwines the worlds of research and practical software development.

In this book, researchers and practitioners report on their results concerning new Software Engineering methods, languages and tools. Topics concerning SE educations are covered as well. The book consists of twelve chapters organized into five thematic parts.

The first part regards Model Driven Engineering (MDE). In Chapter 1, Anna Derezińska from Warsaw University of Technology shares their experience in teaching graduate students model transformation with the QVT language. In the following Chapter 2, Wiktor Nowakowski et al. from Warsaw University of Technology introduce the concept of requirements-level programming. They present syntactic constructs of a requirements language (RSL) and show how they can be interpreted as programming constructs.

In the second part of the book, different aspects of mobile applications development are discussed. Chapter 3 concerns software architectures for mobile applications. Andrzej Zalewski et al. from the Warsaw University of Technology describe a new approach to facilitating the analysis of mobile software architecture. The approach focuses on resolving design tradeoffs by analyzing a set of basic architectural decisions. In the following Chapter 4, Marcin Daszuta et al. from the Lodz University of Technology examine three approaches to the simulation of emotions of Non-Player Characters in mobile games.

The third part discusses some new results concerning the analysis of software code. In Chapter 5, Michał Wroński from the Rzeszow University of Technology investigates

if the famous Zipf's Law for natural language applies to compiled software code in Java, C/C++, and C#. In the following Chapter 6, Grzegorz Kochański from Smart4Aviation, Gdansk Science & Technology Park presents a Java code analyzer written in Prolog that reads bytecode and allows to build directed acyclic data flow graphs.

The fourth part of the book regards non-functional aspects of software systems. In Chapter 7, Aneta Poniszewska-Marańda et al. from the Lodz University of Technology discusses selected aspects of security mechanisms in the Bitcoin cryptocurrency network. Chapter 8 concerns performance of web applications. Radosław Markiewicz and Ziemowit Nowak from the Monterail.com sp. z o.o. and Wrocław University of Science and Technology share their observations on the relationships between various elements of single-page web systems and their user-perceived performance. Finally, Chapter 9 concerns real-time systems. Andrzej Turnau and Wojciech Zwonarz from the AGH Akademia Górniczo-Hutnicza shows that a very inexpensive hardware such as the Raspberry PI v2.0 microcomputer could be used for real-time image processing and control the robotic arm of the Mitsubishi RV-2F robot.

In the last fifth part of the book, the contributing authors discuss different aspects related to networks, distributed systems, and cloud computing. In Chapter 10, Wiktor Daszczuk et al. from the Warsaw University of Technology presents a preprocessor called Rybu, which allows specifying Integrated Model of Distributed Systems using an imperative programming-like input language with a syntax similar to a subset of the C language. In the following Chapter 11, Marek Bolanowski and Andrzej Paszkiewicz from the Rzeszow University of Technology discusses different architecture decisions that allow creating an application platform supporting management of a distributed heterogeneous environment of complex networks. The last Chapter 12 concerns building cloud-based IT systems. In particular, Mateusz Kut et al. from Boehringer-Ingelheim and Rzeszow University of Technology share their experience from designing and implementing a cloud-based vehicle managing system using the Microsoft Azure cloud environment and Raspberry Pi platform.

Finally, we would like to express our gratitude to the contributing authors, to the reviewers for their valuable remarks and to PTI for its continuing support of software engineering publications.

Piotr Kosiuczenko
Lech Madeyski
Mirosław Ochodek
Andrzej Paszkiewicz

Part I
Model Driven Engineering

Chapter 1
Experiences in Teaching Model Transformation with the QVT Language

1. Introduction

Model transformations are essential activities within Model Driven Engineering (MDE) [1]. Different approaches and various languages are used for model transformations [2], [3]. Query/View/Transformation (QVT) [4] is an OMG standard language for specifying model transformations. It contributes as one of core standards to the Model Driven Architecture (MDA). Despite several industrial applications [5] QVT is not widely used due to a lack of knowledge and partially due to a still unsatisfactory tool support. Deficiency of educated professionals is one of obstacles in further application of model-based approaches in practice.

Within teaching of MDE fundamentals there is much effort devoted to modeling with UML and to transformation of models into different forms, like program code, data base schemata, test scripts etc. Transformations to be applied are mainly prepared by developers of CASE tools, or similar experts. However, basing on this, further teaching of software modelling should cover the core technology with two main directions: metamodeling and model transformation, as stated by Bezivin [6].

A model transformation can be implemented in a general purpose language, e.g. Java, hence application of a known notation could have an advantage in education. In this paper, we focus on another approach, in which transformations are developed using a specialized language, namely QVT. It has been shown that despite a new notation and limited time, students were able to practically develop simple QVT transformations.

Since 2008, a course of Advanced Methods for Software Engineering, in short AMSE, has been delivered in our Institute. The course is devoted to development of high quality software. Among others, it includes a module aimed at selected aspects of model engineering. Course modules are accompanied by laboratory classes. In a practical part of the MDE module, model transformations are developed by course participants. An approach based on the QVT language has been applied in the MDE laboratory since 2011.

This paper focuses on education of model transformation, with the stress on independent creation of meta-models as well as development and testing of model transformations written in the QVT language. We share some experiences in teaching general issues of meta-modeling and model transformation. We report about mastering practical skills in model transformation with QVT.

The paper is organized as follows. In the next Section, the QVT language is briefly introduced. We give basic information of the AMSE course in Section 3. Section 4 reports on laboratory classes with QVT and experiences gained. Assessment of other MDE issues is presented Section 5. We discuss related work in Section 6 and conclude the paper in Section 7.

2. Preliminaries of transformation with QVT

Query View Transformation Language (QVT) is a hybrid declarative/imperative language devoted to transformation purposes [4]. It is specified and maintained by OMG in cooperation with many organizations and companies. Its concise overview can be found in a Kurtev's paper [7].

QVT is specified in conformance to the MOF (MetaObject Facility) standard [8], which is the OMG base specification in the model-driven engineering. MOF provides a meta-meta model that conforms to itself. It is also used as a reference model in various kinds of models and modelling approaches, including QVT transformations.

QVT architecture consists of a Core language, Relations language, and Operational mappings. QVT Core is a simple declarative language. Typically, it is not used independently but incorporated in fundamentals of other QVT facilities.

The Relations QVT language is used for declarative specification of the relationships between models. A transformation between models is described by a set of relations that must hold. Models conform to their model types, which are meta-models conforming to MOF.

The Operational QVT is an imperative language in which bidirectional transformations are to be written. Structure of QVTo programs is based on Operational mappings that can invoke other mappings.

In general, we can write operational transformations following an imperative approach, or combine relational transformations with imperative operations. It is typical that the same issue can be implemented in very different ways. Other advantages of QVT are its clarity and easiness of a transformation creation, which writing in traditional languages would take a longer time and more code lines. Moreover, modification of a transformation or its extension appears to be straightforward.

3. AMSE Course Outline

Main goals of the Advanced Methods for Software Engineering (AMSE) course are to: (i) make students aware of the need of assuring high quality software, (ii) comprehend by them selected methods of high quality software production, (iii) enhance skills in certain fields of design and implementation of high quality software.

The course syllabus covers software metrics, software refactoring including refactoring to design patterns; selected methods of software testing, e.g. testing with mock objects; aspect oriented programming, model-driven engineering, and design of formal specifications. The detailed subjects of the course may slightly vary in dependence of a course realization.

3.1. Course Participants and Course Credits

The course is attended by students that hold a bachelor degree of computer science, computer engineering, or an equivalent diploma. They are intended to get MSc degree in computer science. Prerequisites of participants are working knowledge of programming in an object-oriented language (such as C++, Java, C#) and basic knowledge of UML. During the first degree education, the students have attended a basic course on Software Engineering, including a project on modelling and design using the UML notation. Most of the students have developed a simple application based on transformation of UML models to code, e.g. Java. Apart from their educational background, almost all of the students have been working and gained some professional experience, mainly in programming using different general purpose languages.

The AMSE course is an elective course. Students have to select several such courses, according to their interests, specializations, or domains of their theses to be prepared. They are also obliged to collect a certain amount of ECTS (European Credit Transfer System) points after passing courses specified in the curricula.

The whole AMSE course takes 30 hours of lectures and 15 hours of laboratory classes with a supervisor. A student effort of the course is foreseen for about 120 hours, including lectures, tests, classes in laboratories, but also consultation, preparation of projects and self-study. This is counted as 4 ECTS points.

The MDE issues are only a part of the course, and this module can be assessed for 1.2 ECTS. The MDE unit contributes approximately to 36 hours of a student effort. It has been estimated as 8 hours of lectures and a final test, about 3 hours of a lecture preparation and consulting with a supervisor, about 3 hours preparation for the final test, and 4 obligatory hours in laboratory. The remaining 18 hours are aimed at maintaining laboratory tasks, in particular, making familiar with the subject and laboratory instructions, arranging a tool support in an own computer, realizing a project in laboratory and at home, completing a final report, etc.

3.2. MDE Part of the Course

Basic modeling concepts with meta-model ideas and examples are discussed during lectures on MDE. Intuitive and formal definitions of model concepts are presented. Students learn about relations between models and meta-models in a model hierarchy, such as representation and conformance. Different examples, based on simple UML and other IT areas are discussed. Students are provided with the basic knowledge about MOF standard and selected meta-model examples from the UML specification. Different mechanisms of UML extension with and without meta-model modification are discussed. Principles of a profile creation and its application are illustrated with examples. The fundamentals of MDA with its viewpoints, platforms, and model levels are presented.

The rudimentary notions of model transformation are surveyed, together with basic transformation feature. Several approaches to transformation realization and transformation languages are examined and illustrated by simple examples [2], [3]. The examples refer to relational and operational QVT, but also other approaches like model to text solutions, graph-based transformations, etc.

As for a selected transformation language, the most important features of the QVT Relations Language and Operational Mappings presented. Examples cover the basic language constructions, such as a transformation structure, operation mapping, *when* and *where* closures, declarations of objects, working on collections, passing of parameters, writing of helpers and query, usage of the *self* keyword, control statements and loops.

The final section of the MDE lectures is devoted to merging of models, treated as a special kind of transformations. Basic rules and common problems are reviewed and illustrated with examples of languages from Epsilon [9], which is a set of languages and eclipse-based tools supporting m2m transformation and other processing of models.

Overall, the lectures give a broad overview of transformation approaches and languages, while selected notions are discussed in more details. An important part is devoted to QVT, although it is presented as one of many alternatives and compared to other languages.

Apart from theoretical knowledge, the course is aimed particularly at the development of advanced practical skills. The following educational goals have been appointed to be undertaken in the MDE part of the course:

A. comprehension of prepared meta-models,
B. modification (e.g. extension, adaptation) of a given meta-model according to certain guidelines,
C. evaluation of a model conformance to a given meta-model,
D. design and creation of a meta-model based on a given domain description,
E. creation of a meta-model derived from a representative set of models,
F. comprehension of transformations prepared by other parties,
G. usage of transformations in model-driven engineering,
H. evaluation of transformations in conformance to meta-models of an input/output model,
I. modification of a given transformation,
J. design and implementation of a transformation that completes a specified task using source and target models conforming to the same or different meta-models,
K. inspection and testing of a given transformation, design of representative models to test a transformation in regards to meta-models of source and target models.

The number of participants in each course edition was not very big, ranging from 11 to 27 (Table 1.), therefore, it was a chance of an active involvement of students also during lectures. They asked questions, compared different approaches, discussed and solved simple examples. However, those activities referred only to a part of students who were more interested in the subject. Nevertheless, all students were obligated to attend laboratory classes during which above goals have been taken on.

Table 1. Evaluation results of MDE part

Year	**Number of participants**	**Laboratory tasks**			**Final test**		
		Mean	Standard dev.	Median	Mean	Standard dev.	Median
2011	13	7.9	1.9	8.1	5.4	2.0	4.4
2012	14	8.3	2.7	8.8	6.8	1.8	7.1
2013	26	9.3	0.4	9.4	4.3	2.9	2.1
2014	21	6.8	1.9	6.7	4.6	2.5	3.6
2015	11	8.9	2.5	9.4	3.9	2.6	3.1
2017	24	8.6	1.8	9.4	3.7	1.6	3.3
2011-2017	109	8.3	1.7	8.3	4.1	2.1	3.7

4. Laboratory on Model Transformation

One of the general goals of the laboratory classes was demonstrating of the advantages and disadvantages of a transformational approach. We focused on an illustration and independent realization of a model-to-model transformation process.

In laboratory tasks, operational QVT was used. In general, it was simpler to comprehend than relational QVT, as students had been more used to write programs in imperative languages and scarcely in declarative ones.

Prerequisites of model transformation were comprehension of meta-modeling concepts. Therefore, most of laboratory classes started with an introductory test revising meta-modeling concepts and putting them into praxis. Three kinds of main activities were performed during laboratory classes:

- Introductory test on meta-modeling
- Tutorial - preparation of a given meta-model, a given transformation, and representative test models
- Independent realization of a whole m2m process for a given problem.

4.1. Laboratory Environment Setup

Considering tools for MDE, and especially transformation in QVT, availability of tools is much smaller than for UML. Starting in 2010, we selected a set of the following most mature tools: UML ModelTransformation Tool (UML-QVT) v.0.8, SmartQVT v.0.22, Eclipse M2M, QVTParser, and Borland Together. The tools were compared taking into account the following aspects: supported transformation languages, formats of model saving, software license, application type (stand-alone, plug-in), year of an origin, an organization/company managing it, a project web page, a framework or an operating system. Moreover, after performing a set of experiments with the mentioned tools, they were subjectively assessed by a user in two categories: project maturity and GUI friendliness. Evaluation results of the consecutive tools mentioned above, except Borland Together, are the following: project maturity (3, 6, 5, 4) and GUI friendliness (3, 6, 4, 7), where an 1-10 scale was assumed.

This comparison showed that at that time, the best tools for QVT transformation were associated with the Eclipse development platform. During a preliminary course, transformation tasks with the voluntary students were performed using SmartQVT. However, this project was abandoned and some members of the development team jointed the Eclipse M2M community. The Eclipse M2M engine was also used in the Borland Together.

Therefore, since 2011 we have been using Eclipse with the Eclipse Modelling Framework and Operational QVT SDK during the regular AMSE laboratory classes. The Eclipse versions followed the currently available environment, starting with Helios in 2011, until Mars. We kept on with the M2M changes, introducing slight updates to an installation and usage instruction delivered to students, if necessary. Although conceptually the main flow of the process used during tutorials and further student tasks has remained the same.

In teaching of basic modeling with UML, it is often posed a dilemma about using professional but heavyweight tools versus simple ones. Application of a simple tool,

especially during preliminary courses, gives an advantage not to waste too much attention on the technological issues and concentrate on the main concepts. Here, advanced students were more willing to work with commonly used tools, or based on common, familiar interfaces. They tend to prefer professional tools, which are likely to be faced at their work, or at least learning of skills that have a direct impact on their current or future job.

4.2. Laboratory Workflow

Laboratory classes consist of two meetings lasting two hours each. Before the first meeting, all lectures concerning MDE subjects are delivered. Moreover, students are supplied with materials explaining laboratory tutorials, a way of working with meta-models and QVT transformations in a selected environment. During the first meeting, optionally after a short introductory test, a tutorial is realized .

Later, proper tasks to be prepared are distributed. One or two transformations are to be designed, implemented, and tested by each student. Transformations are based on the meta-model used during the tutorial and/or other specified meta-models. The tasks are similar, but as a rule different for each student.

Students have one or two weeks before the next laboratory meeting. In this time, they can prepare their solutions using their computers (an instruction how to prepare and use the environment is given in the laboratory materials). In the meantime, they can also work in the laboratory in their spare time without a supervisor.

During the second laboratory meeting, students have time to complete and improve their solutions. They present their meta-models, runnable transformations, tests of models, and transformed models to a supervisor. They are asked to improve inconsistencies or incompleteness in transformations, if any detected. It is very often, that it is necessary to create more complete test models to test a transformation. For example, models used for testing are intended to cover all element, artifacts and relations, of meta-models of concern.

After presentation and discussion of results, all prepared artifacts with the final documentation are provided to a supervisor to a final evaluation.

4.3. Laboratory Instruction

A laboratory instruction includes detailed guidelines how to use a tool in the laboratory and how to install it in own computers. Moreover, a detailed tutorial is also given, covering building of a given meta-model, installing it as a plug-in to the environment, verifying whether the meta-model extends the environment, creating a model consistent with the meta-model, development of a given transformation, and running of the transformation on a test model. In the tutorial, all steps and all selected options, included data, possible messages, etc. are exactly specified.

Apart from the instruction, during the tutorial realization (Sec. 4.4) students were assisted by a supervisor and cooperated with each other to clarify any doubts.

Students were also provided with a concise QVT description. It was based on the QVT specification but concentrated on the most important language features. Students could also make use of some examples that showed typical application of the language.

4.4. Laboratory Tutorial and Tasks

Students have to complete an MDE process staring from creating a meta-model, writing a transformation working on the meta-model, building test models and testing the transformation. During the tutorial phase, a meta-model and a transformation given in the course materials are used. The main goal of this first laboratory meeting is to make them familiar with the process realization, tool support, practical application of MDE concepts, running of transformations and preparing of test models.

The tutorial task was based on a simple class meta-model. A package can contain other packages and classes. A class has a name, and other characteristics. It can contain other classes, attributes, and methods. A simple tutorial transformation changes names of classes. It works and creates an output model, but on purpose it does not preserve a structure of aggregations among packages and classes. With such a basic example, students learn to apply different models to test transformations and reveal their weaknesses.

Furthermore, a draft of another meta-model was given that describes a university domain familiar to students. For example, many persons constitute a crowd, a person could be a student, a student attends a university, a person has documents, student has index with marks, etc. Based on this a participant had to build a meta-model that covered at least all necessary elements and relations to complete a given task, such as:

- *Transform a person with valid documents and of a given age into a student associated with a certain university and provided by a student identification number.*
- *Transform each student fulfilling given formal requirements and having a desired average mark of all courses into a graduate student with a bachelor degree.*

4.5. Laboratory Challenges

Students were encouraged to complain and assess the tool support during the whole process. Most complaints were focused on preparation and application of meta-models, partially because of an insufficient tool support in verification and detection of errors in meta-models. Errors of this kind were sometimes manifested only during preparation or running a transformation. In some cases a model creation crashed giving no reasons why. Any visible mistakes have been discovered in a meta-model, hence the fastest solution was to rewrite it once again. One the other hand, students observed that preparation of a proper meta-model is an activity performed once, or rarely, whereas transformations can be written many times for a given meta-model.

Transformation programs were not very long, but support for identification of errors was also not sufficient. In QVT it is allowed to write some functions on collections that has no sense. A programmer is not warned about it although types of collections are defined. The basic criteria of test case generation were coverage of meta-model concepts, contract specification of transformations, and coverage of transformation code. Students could observe insufficiency of use only ad-hoc test models instead of a set of representative models covering all concepts of meta-models. However, development of comprehensive test models can lead to many test cases or complex test models in case of a complicated meta-model. Students complained about a lack of support for creating and running unit tests for QVT.

QVT has a flexible syntax, in which typical notations originated from different commonly used programming languages are accepted. For example: comparison operator as "=" or "==", checking of inequality with "<>" or "!=", comments limited by /*..*/, //…, or --, etc. These are details, but altogether have made the language easy to use by students who are not spending much time to study it but try to write code based on representative examples and provided basic language principles. This is true under an assumption that those students have already been familiar with general purpose languages like C, C++, Java or C#, have been acquainted with the SQL notation, and even have tackled some specification notations, like OCL, Alloy, or functional languages as Haskell, etc.

An important observation is that running of a transformation is similar to running another program, which is a well-known activity for students. They noticed a strength of the language and pointed out that problems originated from different domains can be solved with the same transformational notation. They were also aware of potential opportunities for adopting transformations to their professional needs.

Students were often positively surprised that a concise and meaningful transformation code can accomplish required tasks. After having evaluated correct transformation code of their tasks, some students made the code refactoring finding solutions that were well designed and more compact.

5. Assessment of the MDE Module of the Course

There are four main sources of student contributions used in an assessment of the MDE part of the course:

1. Student activity during lectures and tutorials (0-2 points)
2. Introductory test during laboratory classes (0-2 points)
3. Tasks prepared, run and documented in labs (8-9 points)
4. Tasks solved during final tests (4-8 points per task)

Given points are used for calculation of a final course score, whereas there are 60 points for the whole course.

Before starting a laboratory class, an introductory short tests can be performed. Tests of this kind have a low impact on the overall evaluation, as they were intended to revise basic concepts of meta-modelling before starting their practical utilization. Two exemplary tasks are given below.

A. *Draw an exemplary meta-model to which conforms a given model A (***Figure 1***).*
B. *For a given meta-model X (***Figure 2***) describe a set of models that conform to it. Does model B conform to X ? Justify the answer.*

In the task A, although a possible solution is conceptually very similar to those discussed during lectures, the first attempt to design a meta-model caused several problems. In general, it is much more straightforward to use meta-classes for named elements and graph nodes, e.g. classes, their attributes, than for unnamed relations.

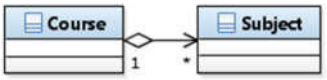

Figure 1. Model A

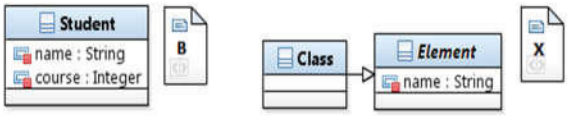

Figure 2. Model B and meta-model X

In the final tests, tasks referring to the MDE part have covered two main areas. One category of tasks deal with different forms of models and meta-model design, e.g.:

- creation of a meta-model, while having a model expressed as a meta-model instance,
- verification of a model conformance to a given meta-model,
- finding of inconsistencies, missing elements in a meta-model; designing an extension of a given meta-model.

Second type of tasks concerns creation and inspection of a transformation written in different notations, mainly QVT, for example:

- for a given meta-model and transformation, explain the meaning of particular statements/procedures in the context of meta-model elements,
- extend a given transformation with an additional helper or mapping that is used in the transformation,
- complete a given transformation, in which missing statements are denoted by code gaps,
- write a simple transformation using a given meta-model(s).

Examples of tasks to be solved during a final test are given below:

A) For a given meta-model *K1* (**Figure 4**) and model *Job* (**Figure 3**) specify:
   - all relations between model elements and corresponding meta-model components,
   - all missing elements of the meta-model
   - other inconsistencies of the meta-model

   Improve the given meta-model, assuming the model *Job* is correct.

B) Extend the improved meta-model (result of point A) in order to specify with named stereotypies such elements as *Employer, position, define_Rate*.

C) Develop a transformation of models that conforms to the extended meta-model (result of point B). The transformation adds a suffix 'S' to names of model elements that have stereotypies. The transformation has to be written in the relational or operational QVT. All statements have to be commented.

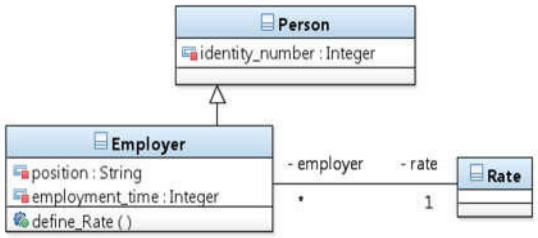

Figure 3. Model Job

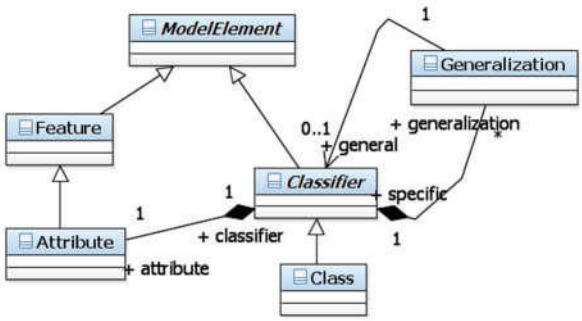

Figure 4. Meta-model simple class K1

During a final test, students are allowed to use additional reference materials, lecture notes, especially manuals specifying syntax and semantics of the QVT language. They are not allowed to communicate with each other or to surf on the Internet.

At the beginning of the MDE classes, typical problems faced by students correspond to comprehension of fundamental meta-modeling ideas, an application of meta-models in a conscious manner, ability of discriminating between different levels of concepts, e.g. of current models, meta-models, meta-meta-models. Those problems were partially overcome after laboratory classes.

The following errors in the domain of meta-modeling encountered in final tests:

- mixing of concepts from a model and meta-model level (17%),
- incomplete identification of conformance relations between model elements and corresponding meta-model entities (17%),
- detection of not all missing elements of meta-models (36%),
- correction of meta-model mistakes in a wrong way (34%),
- incorrect incorporation of a stereotype into a meta-model (20%)

The number denotes a percentage of tests in which a mistake of this kind encountered. The following types of errors in QVT transformations were observed in final tests:

- improper reference to meta-elements (10%),
- fault in domain of an operation (10%),
- missing part of a transformation, e.g. mapping, handler (10%),
- lack of a transformation code (10%),

Results of the MDE part of the course are summarized in Table 1. The results are given for the laboratory and final tests separately and normalized to 10 points maximum.

In general, scores of both contributions highly depended on a student attitude towards the subject. In case of the laboratory task, another crucial factor was a student background.

Most of the students who completed laboratory tasks with at least satisfactory notes (i.e. obtained above half of points from the possible maximum number) coped fairly well with tasks from a final test. However, it was much easier to solve a problem, when an executable result could be visualized as an output model. Some students have difficulties in applying their knowledge in an abstract manner. Moreover, some tests, e.g. in 2015, were written before finalizing laboratory classes, which resulted in lower test results.

6. Related Work

There are many references to teaching of object-oriented analysis and design of systems with the UML notation, e.g. [10], [11], as this is a fundamental subject in the software engineering area. Therefore, selection of tools to be used in education is also discussed mainly in the context of UML modelling and design, as well as application of build-in transformations in MDE [12]. Some experiences are also reported on advanced courses covering various aspects of Model-Driven Development [12]-[18]. However, there are no many reports on educational issues of the direct model transformation. Transformations are often applied in scripting languages, as PHP [18] or general purpose languages, as Java [19]. To the best of our knowledge few of them have reported teaching experiences in developing model transformations in specialized languages [12], QVT in particular [5].

Batory &Azanza discussed problems in using Eclipse Modeling Tools in MDE education, both in respect to installation and tool usage [19]. A version of Eclipse was eventually posted that had all tools installed. It was assessed that only 25% of the upper-division undergraduate class got their task right, 50% had mediocre results, and the remaining gave up the task. After an additional time supported with tutorial help, 80% of students completed right the task. Based on these experiences they developed a new MDELite tool that intends to overcome those inconveniences [20]. There are no enough details to thoroughly compare this case to the ours, and the approach was focused on different subjects, namely relational databases. During our laboratories, we also used Eclipse tools. Students were supported by very detailed instructions prepared by other students and a supervisor. In general, we do not have faced problems with a software installation. The main problems were comprehension of basic concepts and recognizing of any benefits of a transformation usage. Postgraduate students that participated in our courses might be more experienced, including their work in IT area. The EMF environment was also used during the MDE course presented in [14].

An interactive approach to an MDE process combined with teaching of program translation techniques was presented in [12]. The whole course required a much bigger student effort than presented in this paper, hence transformation tools were prepared by course participants. It was emphasized the necessity of practical application of presented methods and trained skills. Although reporting about the profitable and prosperous course, the authors mentioned unpopularity of MDE among students.

A similar practical approach was described by Schmidt et al [18]. An MDE course was focused on creation a simple tool for code generation from UML models. No domain

languages, such as transformation ones, are considered, as the whole process try to use tools already known to students.

A whole comprehensive MDE course is also discussed in [21], although no details about teaching QVT are given. The authors faced difficulties with concept comprehension, meta-modeling in particular, using new languages (QVT including), and insufficient tool support.

Brosch et al. reported their experience in delivering an advanced modeling course [16]. This course partially covered the similar subjects as the MDE part of AMSE, but the whole course was devoted to model engineering, therefore, more class units covered meta-modeling, and transformations were discussed with more details. Students completed laboratory tasks, during which they created meta-models with Eclipse Modeling Framework (EMF), defined constraints in OCL, developed an ATL transformation, and performed code generation to SQL and Java. In general, the authors were satisfied with their results and have similar experiences to ours, although complained about immature MDE tools. However, neither during the lectures nor in practical tasks they did use any type of QVT.

A common problem in modeling evaluation is an assessment of a model quality. It is partially done by a manual inspection of a supervisor, which is subjective and laborious. Application of an executable modeling language with continuous validation helped to solve this problem in a modeling course [17]. However, using a model transformation, students can directly run their solutions and observe the results. The main validation problems were moved over to the creation of representative test models. Testing of transformations is surveyed in [23].

A quality model of a QVT transformation is presented in [5]. Authors discussed a set of best practices, quality metrics, as well as a tool support for the model transformations in QVT, but also AMT (with the similar constructs to QVT) and Java (the mostly used for model transformation). The quality model is an interesting proposal, especially for big industrial transformations. In small transformations developed during an introductory course, only a subset of the quality properties could be applied. Testing of transformation can be improved by the QVTo code coverage support that was implemented for Eclipse [5]. It would be promising to incorporate these approaches into the future AMSE courses.

It is characteristic that novice developers tend to apply language patterns known from other programming languages, which results sometimes in good but sometimes in improper solutions. Similar was reported in [5], where reproducing of GPL program structure caused using a large *init* sections inside a QVT mapping, instead of a population section.

7. Conclusions

In this paper, experiences of introducing model transformation with QVT into an advanced software engineering course are presented. Basing on student knowledge of other programming languages and an extended tutorial, it was possible to successfully complete tasks with a new language in relatively short time. One of the most important obstacles in education of MDE issues was convincing students of any benefits of the transformation approach.

Teaching of transformation in general, illustrated with QVT applications in particular, could be thoroughly extended with more systematic and tool supported specification, verification, and design patterns. However, a still limited usage of transformations and their specialized languages in an industrial praxis made students reluctant to a deeper study on these subjects, similarly to education about model-driven development or formal specifications. There is also an open issue whether it is more suitable to use a domain language, like QVT for a model transformation, or a general purpose language like Java.

Acknowledgments. The author would like to thank P. M. Zajączkowski for preparing a preliminary version of the instruction for the QVT laboratory within the AMSE course. Preparation of lecture presentations for the course was partially supported by EEA Norway grant in project of Education System Development Foundation.

References

[1] S. Sendall, W. Kozaczynski, Model transformation - heart and soul of model-driven development, IEEE Software, 20(5), (2003), 42-45 doi: 10.1109/MS.2003.1231150
[2] K. Czarnecki, S. Helsen, Classification of Model Transformation Approaches, In: Proc. of the 2nd Workshop on Generative Techniques in the Context of Model Driven Architecture (co-located OOPSLA'03), (2003)
[3] T. Mens, P. van Gorp, Taxonomy of model transformation, Electr. Notes in Theoretical Computer Science, Vol. 152 , March 06, (2006), 125-142
[4] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. http://www.omg.org/spec/QVT/
[5] C. M. Gerpheide, R. R. H. Schiffelers, A. Serebrenik, Assessing and improving quality of QVTo model transformations, Software Qual. J. 24(3), (2016), 797-834, doi 10.1007/s11219-015-9280-8
[6] J. Bezivin, Teaching Modeling: Why, When, What? In: Ghosh, S. (ed.): MODELS 2009 Workshops. LNCS, vol. 6002,. Springer, Berlin Heidelberg (2010), 55–62, doi: 10.1007/978-3-642-12261-3
[7] I. Kurtev, State of the art of QVT: a model transformation language standard. In: Schürr, A., Nagl, M., Zündorf, A. (eds.), AGTIVE 2007, LNCS, vol. 5088, (2008), 377–393, doi: 10.1007/978-3-540-89020-1_26
[8] OMG, Meta Object Facility (MOF) Core Specification. http://www.omg.org/spec/MOF/
[9] Epsilon - http://www.eclipse.org/gmt/epsilon/.
[10] J. Garzas, M. Piattini, Improving the teaching of object-oriented design knowledge, ACM SIGCSE Bulletin, 39(4):108 (2007)
[11] P. J. Burton, R.E. Bruhn, Using UML to facilitate the teaching of object-oriented systems analysis and design, Journal of Computing Sciences in Colleges,19(3), (2004), :278–290
[12] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, Tool use in software modelling education, in Proceedings of the Educators' Symposium co-located with ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS2013) (2013)
[13] J. Poruban, M. Bacikova, S. Chodarev, M. Nosal, Pragmatic Model-Driven Software Development from the viewpoint of a programmer: teaching experience, In: Proc. of the 2014 Federated Conference on Computer Science and Information Systems, ACSIS vol. 2, (2014), 1647-1656, doi: 10.15439/2014F266
[14] P. J. Clarke, Y. Wu, A. A. Allen, T. M. King, Experiences of Teaching Model-Driven Engineering in a Software Design Course, In: ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems, MODELS'09. Denver, Coloardo, USA (2009)
[15] A. Derezińska, P. Ołtarzewski, Evaluation of the Contract-Aware Software Development Process in a Controlled Experiment. In: Elleithy, K., Sobh T. (Eds), New Trends in Networking, Computing, E-learning, System Sciences, and Engineering, LNEE vol. 312, Springer, (2015), 365-372, doi: 10.1007/978-3-319-06764-3_45

[16] P. Brosch, G. Kappel, M. Seidl, M. Wimmer, Teaching model engineering in the large, In: Proc. of 5th Educators' Symposium in conjunction with the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2009). Denver, CO, USA (2009)
[17] H. Burden, T. Adawi. Mastering Model-Driven Engineering, In: Proc. of ITiCSE, (2014), 351-351. doi: 10.1145/2591708.2602665
[18] A. Schmidt, D. Kimmig, K. Bittner, M. Dickerhof, Teaching Model-Driven Software Development: revealing the great miracle of code generation to Students, In: Proceedings of the Sixteenth Australasian Computing Education Conference (ACE2914), Auckland New Zealand (2014) 97-104
[19] D. Batory, M. Azanza, Teaching Model Driven Engineering from a relational database perspective, Software and System Model. (2015), doi: 10.1007/s10270-015-0488-7
[20] MDELite. https://www.cs.utexas.edu/users/schwartz/MDELite/index.html
[21] A. Hamou-Lhadj, A. Gherbi, J. Nandigam, The Impact of the Model-Driven Approach to Software Engineering on Software Engineering Education, In: Sixth International Conference on Information Technology: New Generations. (2009), doi: 10.1109/ITNG.2009.160
[22] L. Lu´cio, et all., Model transformation intents and their properties, Softw. Syst. Model, 15, (2016), 647-684, doi:10.1007/s10270-014-0429-x
[23] G. M. K. Selim, J. R. Cordy, J. Dingel, Model transformation testing: the state of the art, In: Proceedings of the 1st International Workshop on the Analysis of Model Transformations (AMT), (2012), 21-26, doi:10.1145/2432497.2432502

Chapter 2

Using Precise Requirements Models as Programs

1. Introduction

Constantly rising software complexity combined with hard time-to-market restrictions causes the necessity to introduce new approaches to software development. This includes higher level programming languages which allow program creators to solve the same problem with less lines of easier to understand code. Such languages can be supported by Model Driven Engineering (MDE) [16] which is one of the most interesting and quickly developing approaches focusing on handling of high-level models and transforming them even up to executable code. One of the tools representing the MDE paradigm is ReDSeeDS [6] which offers an open framework supporting a development method based on models consisting of use case scenarios and domain vocabularies. It implements the Requirements Specification Language (RSL) [18], [11] which uses constrained natural language sentences allowing end-users to understand specifications presented as precise requirements definitions. Moreover, the precisely written platform-independent specification allows to translate it directly to code using one of the platform-specific transformations. In other words, the requirements models can be treated as programs.

The RSL-based solution separates essential complexity of the problem domain consisting of business domain rules and application logic, from the accidental (technological) complexity related to platform-specific design and implementation [7]. The complexity of software development process using ReDSeeDS is greatly reduced from the user's point of view. The developer needs only to create a requirements specification expressed in RSL. The whole accidental complexity is encapsulated by the transformation program used to convert the specification into code. Additionally, choosing one specific target platform over another is just a matter of switching to the correct transformation profile.

This paper introduces the concept of requirements-level programming. In the following sections we present syntactic constructs of a requirements language (RSL) and show how they can be interpreted as programming constructs. This means assigning specific runtime semantics to a high-level language which uses constrained natural language and end-user domain vocabulary.

2. Related work

Transition from requirements to design or implementation is considered as a difficult activity during software development. Complexity related to this step can lead to various errors. This is mostly associated with ambiguity of software requirements specifications. To eliminate this problem, some form of constrained natural language can be used. Such notation can help in providing semi-automated ways to generate analytical and design models or even certain code artefacts. There exist some sparse approaches to solve this problem. Some work was focused on requirements and their precision both by defining new languages for this purpose [17] as well as proposing proper use of the existing ones [9]. However, such approaches usually lack any attempts related to the ultimate step of code generation.

Among the existing approaches, some proposes formalising the specification of requirements in the form of use case scenarios. Giganto et al. [10] propose an algorithm to identify use case sentences from requirements specifications written in controlled natural language and as a result – automatically obtaining classes from use cases. Mustafiz et al. [13] propose a set of transformation rules for creating different types of behavioural diagrams from use case scenarios. Deeptimahanti et al. [8] suggest to analyse requirements specifications presented in natural language format using Natural Language Processing techniques and to generate use case and class models. Still, natural language seems too ambiguous and more contrained notations are necessary. On the other hand, it can be argued that more formal textual specifications are often too formalized and thus difficult to understand by the end-users who provide information about their requirements [9].

Solutions focused on code generation often use graphical notations like UML [14] diagrams to specify static and dynamic aspects of systems. One example is an open source AndroMDA [1] code generation framework which supports the Model Driven Architecture [4] paradigm. As input, AndroMDA takes UML models from CASE tools and provides generation of deployable applications and software components.

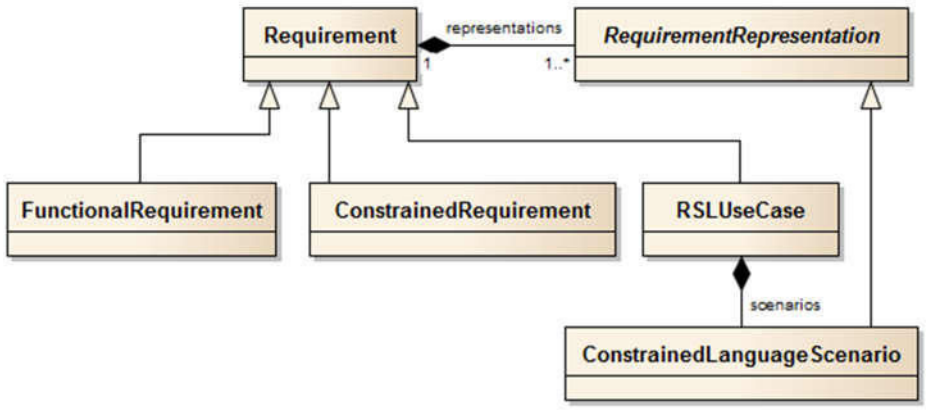

Figure 1. Abstract syntax of the use case model in RSL

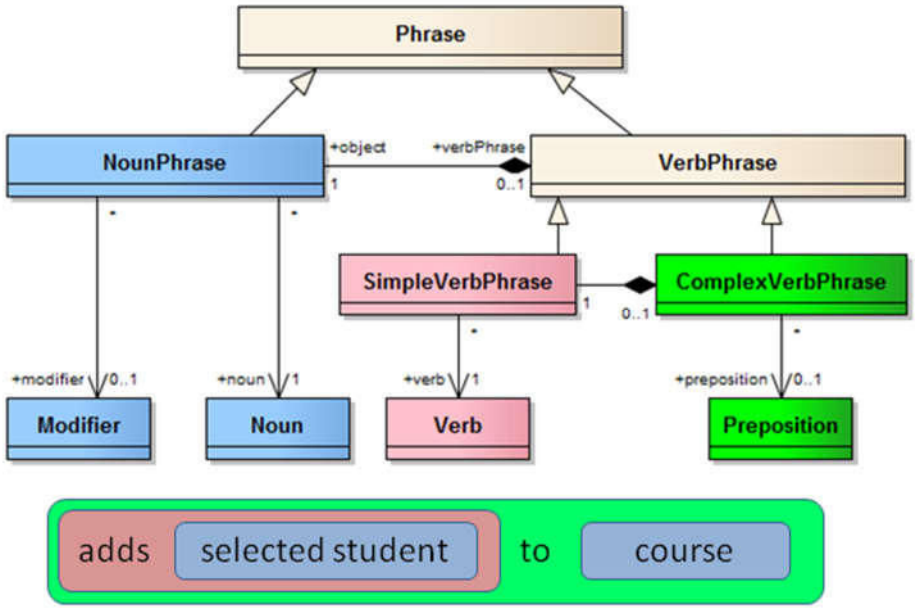

Figure 2. Abstract and concrete syntax for sentences

3. RSL syntax

The Requirements Specification Language (RSL) is defined by a formally specified meta-model. Figure 1 presents the top-level fragment of the RSL meta-model containing the hierarchy for requirements entities. This is the most general part of RSL that provides syntax for requirements and their representations. There are few types of requirements, where some of them are presented in the figure: "FunctionalRequirements", "ConstrainedRequirement" and "RSLUseCases". For the purpose of this paper, emphasis will be put on "RSLUseCases" which are very similar to the widely-known UML use cases. Each "RSLUseCase" represent some part of system functionality described by at least one "RequirementRepresentation" in the form of a "ConstrainedLanguageScenario".

A "ConstrainedLanguageScenario" is a scenario which consists of several numbered constrained-language sentences. The sentences can be divided into two groups. The first one contains sentences which describe interactions between users and systems. Such sentences use a simple Subject-Verb-Object (SVO) grammar[1] . Each SVO sentence consist of two parts - a "Subject" and a "Predicate". A Subject is represented by a "NounPhrase" which can be either an actor or a system element. A Predicate is represented by a "VerbPhrase". Figure 2 presents a fragment of the RSL metamodel for

[1] Note that SVO sentences do not contain articles like 'the' or 'a'

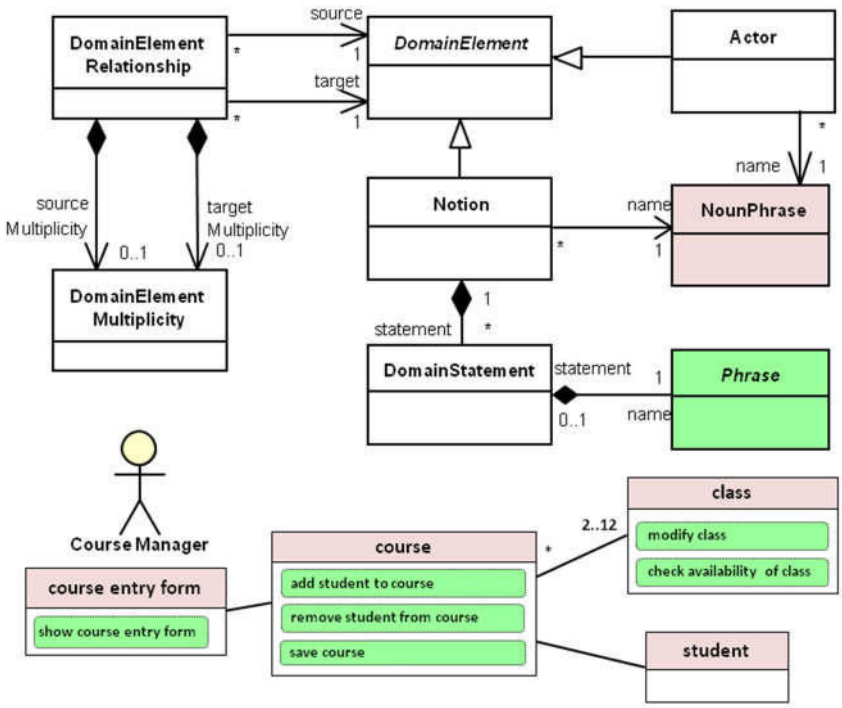

Figure 3. Abstract and concrete syntax for domain elements

phrases. Each "NounPhrase" contains a "Noun" and an optional "Modifier" to more precisely describe the "Noun".

"VerbPhrases" are grouped into "SimpleVerbPhrases" and "ComplexVerbPhrases". A "SimpleVerbPhrase" applies to a simple SVO sentence and consists of a verb and a noun e.g. *adds selected student*. A "ComplexVerbPhrase" applies to a more complex SVO sentence (with an additional object). It extends the "SimpleVerbPhrase" with a preposition and an additional "NounPhrase" representing the indirect object in an SVO-O sentence.

Beside SVO sentences, there are sentences responsible for control flow in scenarios. Such sentences allow for constructing more complex application logic.

- "ConditionSentences" are responsible for forking scenarios into one or more flows. The condition description indicates which scenario to follow during a use case execution instance. Creating a "ConditionSentence" causes creation of an alternative scenario in the containing use case.
- "RejoinSentences" can be used to merge control flows and go back from an alternative scenario to the base scenario.
- "InvocationSentences" allow for invocation (calling) of other use cases (their scenarios) from the scenario level.

If we consider SVO sentences, we can note that they are composed of phrases which it turn are composed of domain elements. RSL introduces a separate notation for domain

models. Every noun occurring in a scenario has its own representation as a notion in the associated domain vocabulary. Figure 3 presents abstract and concrete syntax for such "DomainElements". We can note that they can be divided into three groups: "Actors", "SystemElements" (note that system elements are not presented in the figure) and "Notions". The first two groups refer to subjects in SVO sentences and determine the basic meaning of particular scenario sentences. A separate type of "DomainElements" are "Notions" representing business entities, buttons, windows and other elements occurring in the system to be built. "Actors", "SystemElements" and "Notions" have names in the form of "NounPhrases" but "Notions" additionally contain a set of "Phrases" being "DomainStatements". Such statements can be directly used within scenario sentences, which is illustrated in Figure 4 as compared with Figure 4.

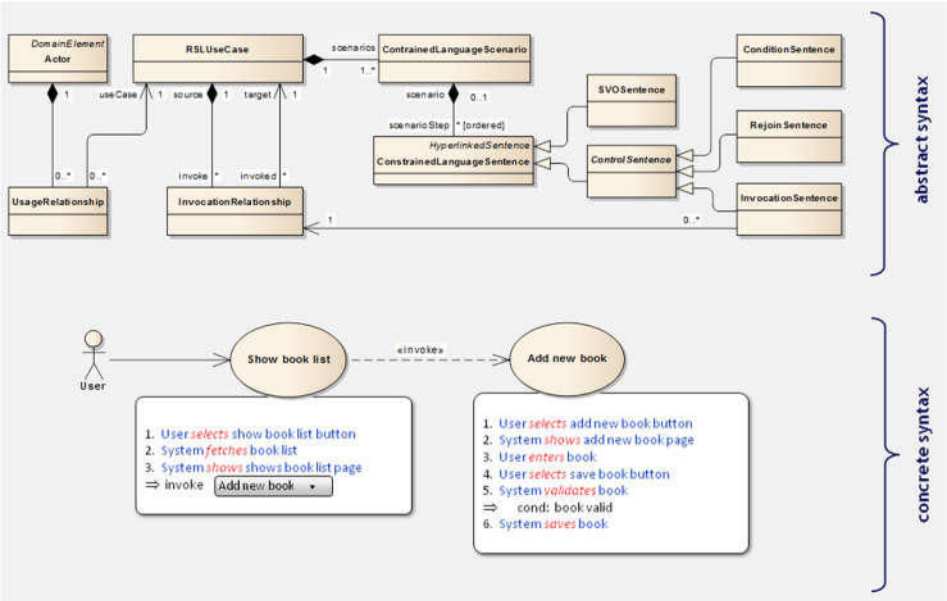

Figure 4. Abstract and concrete syntax for RSL scenarios

Figure 4 presents both the concrete and abstract syntax for use cases and their scenarios. Scenarios are composed of "ConstrainedLanguageSentences" - SVO sentences and control sentences. It can be noted that RSL introduces "invocation" relationships and related "invocation" sentences that mark places where one use case invokes the functionality of another use case. SVO sentences are composed of phrases, as specified in Figure 2. For simplicity reasons we have omitted this part of the meta-model.

4. Specifying runtime semantics for RSL

To be able to interpret RSL as a programming language, we need to give its construct a meaning in the context of computer runtime. Here we will concentrate on explaining the meaning of SVO sentences. For this purpose we identify several different types of

such sentences. The distinction lies in determining flow of control between the participating actor(s) and the system to be built. The direction of this flow of control has a deciding impact on the equivalent code.

- Actor-to-system sentences – represent interactions initialized by a user evoked action e.g. a button press or a link click. Such sentences ere equivalent to code resulting in generation of a triggering element on the currently displayed window.
- System-to-actor sentences – this kind of sentences means that the system is sending a signal to a user by displaying a window (generally: a UI frame element) or some other kind of message. Such sentences ere equivalent to code calling a procedure to show a specific UI frame (window etc.).
- System-to-system sentences – represent an internal data-related operation done by the system like saving, processing or validating. This is equivalent to calling a specific domain (business) logic operation to save, process or validate data.
- Actor-to-actor sentences – such sentences are used for explanatory purposes. They do not have any impact on the system behaviour because they describe actions that don't require any action from the system perspective like e.g. filling a form by a user.

By classifying an SVO sentence with an above classification we can recognize which object is currently in control: the actor or the system. This information allows to interpret other sentences like condition and invoke sentences. Condition sentences occurring when an actor controls the flow mean that the actor decides which path to follow by pressing a specific triggering element (e.g. a button). When the system is in control and a condition sentence occurs, this is equivalent to some "if-else" code fragment inside the application logic code of the system.

To increase precision of our semantics, we also need to determine what kind are the objects in SVO sentences. This can be done by defining additional stereotypes for the "Notion" elements referred-to from the objects in SVO sentences. For this purpose we define the following Notion types.

- **Concepts** are domain (business) entities to be stored and processed by the system. We can also distinguish concepts that are persistent (to be stored by the system) or special "view" concepts that are used for temporary purposes of presenting specific data. In this paper we will not expand this division for brevity. Concepts have several semantic explanations. They are equivalent to data transfer objects, and can be explained through database tables, and business logic classes.
- **Attributes** describe the Concepts' properties just like attributes in UML describe class' properties. In RSL, Attributes are separate notions, and each Attribute should be connected with at least one Concept. Moreover, each Attribute has its specific type which describes specific kind of data it represents like String, Integer, Date etc. The semantics of Attributes is equivalent to e.g. attributes in data transfer objects and columns in derived database tables.
- **Frames** (or Screens) are representations of various kinds of UI frame elements, depending on the target technology. For instance, for web-based technologies Frames will be transformed into web-pages and for desktop technologies – into

windows. If a window is connected with a Concept, its Attributes will be used as basis of window fields. Each string Attribute will be transformed into a label with text field, each Boolean into checkbox etc.

- **Messages** represent simple modal windows used to show some error or confirmation message to the user.
- **Confirmations** also represent simple modal windows with added functionality which allows to answer a question through pressing an "Ok" or a "Cancel" button.
- **Triggers** represent links, buttons or any other elements that can trigger interactions of a user with the system. Just like for Frames, they are platform-independent, and their final form depends on the platform specific transformation.
- **Lists** are used to group many entities of the same Concept into a table or a list. By connecting a List with a Frame we can indicate that the particular list of elements will be displayed as part of the Frame.

The above informally introduced semantic rules for RSL constructs can be further formalised (which we omit for brevity). Having an organised list of translation rules from RSL to code we can develop appropriate transformation programs. This is presented in the next section.

5. Generating code from RSL

The software development and code generation process that uses the ReDSeeDS tool is presented in Figure 5. The first step is to formulate and write requirements in RSL according to the rules described in the previous sections. The tool supports the process by offering a specialized scenario editor which enforces the SVO grammar, and precisely attaches sentence elements with appropriate phrases and notions in the domain model.
The next step is to choose and execute an appropriate transformation. The developer has a choice of transformations that produce code compliant with specific target platforms. These transformations are developed using a dedicated model transformation engine and language. We have chosen MOLA (MOdel transformation LAnguage) [12] due to its clear visual syntax. MOLA is a graphical language which uses patterns matching algorithms at the meta-model level to transform one model into another. In our case we can transform a requirements model in RSL into a UML class model with embedded code text for the class methods.

An example MOLA procedure is presented in Figure 6. This procedure allows to distinguish an SVO sentence (with one object) from an SVOO sentence (with two objects). Each grey box represents a rule. The first rule checks if an "SVOsentence" contains a "Predicate" in the form of a "SimpleVerbPhrase". If not (see the {ELSE} clause), the next rule checks whether the predicate is a "ComplexVerbPhrase". Regardless of the "Predicate" type, the "utl_addProcedureCall" procedure is called but with different parameters which means generation of different code fragments, depending on the sentence type.

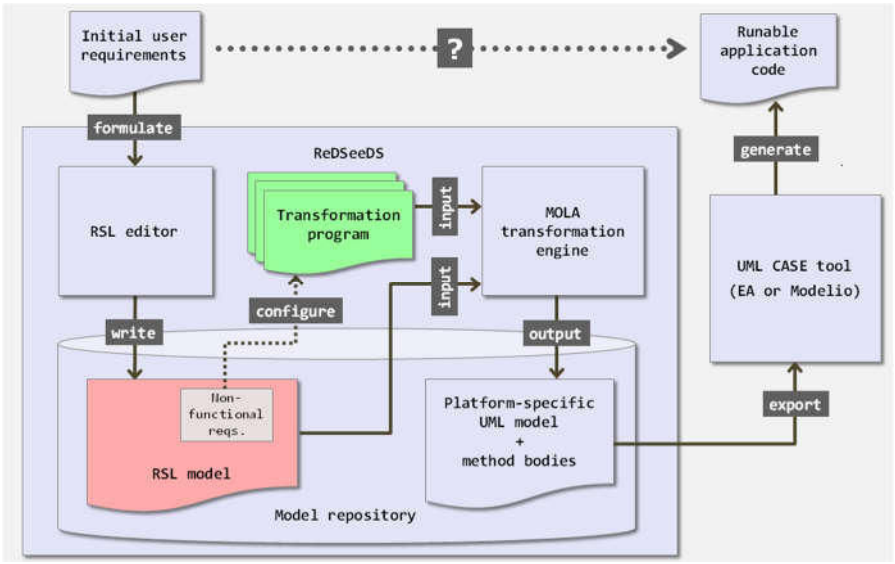

Figure 5. Overview of the code generation process using the ReDSeeDS tool

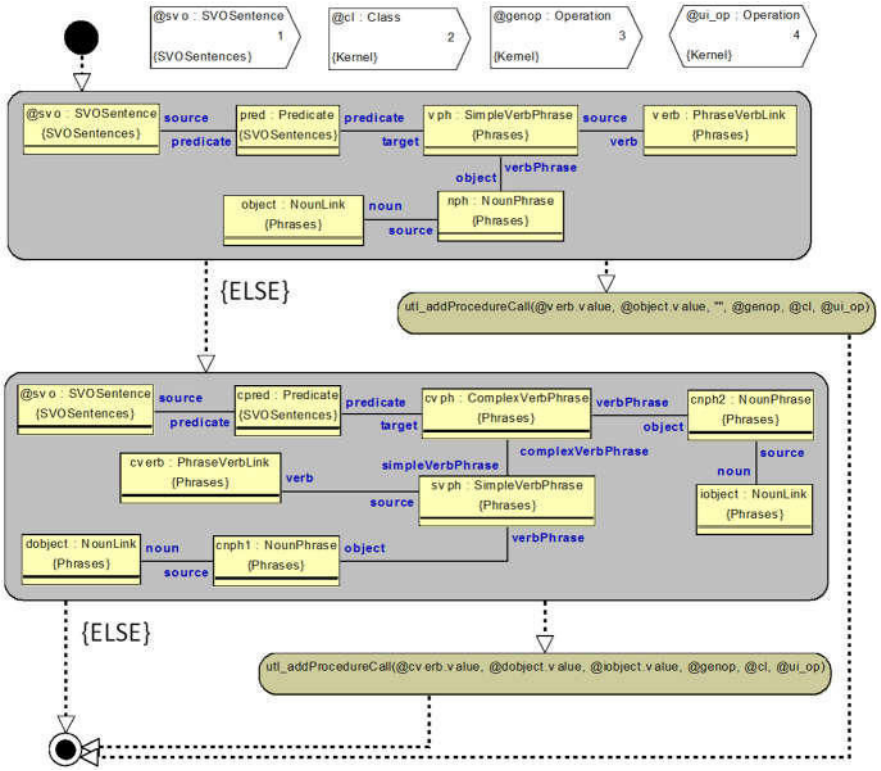

Figure 6. Example MOLA procedure: distinguishing SVO from SVOO sentences

The transformation program analyses scenarios, sentence by sentence, taking under consideration control flow, types of notions attached to the sentence objects and the UI context (what is the currently displayed frame/screen). Depending on the combination of these factors, different code is generated. The following list presents typical combinations and describes the generated code. We assume that the output is a system in the MVP Architecture [15] based on the Echo framework [2]. Most of the system code is automatically generated with complete View, Presenter, and Data access layers and stubs of the Model layer.

- Actor-to-Trigger is type of an Actor-to-System sentence in which the user presses some button and expects a response from the system. This type of sentence generates code to present a button in the class responsible for the currently opened window (in the View layer) and a method responsible for handling an associated button event in Presenter layer. Because it is an Actor-to-System kind of sentence the next sentence's subject should be the system, otherwise the validation mechanism will raise a transformation error.
- System-to-Frame, System-to-Message, System-to-Confirmation are type of System-to-Actor sentences. They generates code in the Presenter layer which calls a method to display a proper method to render an appropriate web-page in the View layer.
- System-to-Concept is a System-to-System kind of sentence where the system does some internal operations on a Concept like saving or validating. In this case the Presenter layer calls a method from the Model layer.
- Actor-to-Concept, Actor-to-List, Actor-to-Frame are types of Actor-to-Actor sentences. As presented in the previous section, such sentences do not cause any code generation. This is because the system does not have to react in any particular way to such user actions. However, these sentences helps the specification become more natural to read.

In summary, the output of the transformation is a UML model consisting of classes with code, attributes and relations. The next step is to export the UML model and generate code with a tool containing a proper code generator. ReDSeeDS currently supports export and code generation using Enterprise Architect [3] and Modelio [5]. It is worth mentioning that the whole generation process from user perspective can be reduced to a choice of transformation. The tool offers functionality to automatically export the UML model and generate source code using Enterprise Architect's or Modelio's API.

6. Illustrative example and student case studies

To illustrate the presented approach to treat RSL specifications as programs, we will use a simple use case model presented in Figure 7 (top). This model contains a single use case with two scenarios. The figure illustrates how particular sentences translate into operations within interfaces of the three layers of the MVP pattern. Actor-to-Trigger sentences (e.g. 'User selects save book button') translate into operations of the Presenter layer (e.g. 'SelectSaveBookButton'). System-to-Frame and System-to-Message sentences (e.g. 'System shows add new book page') translate into operations of the View

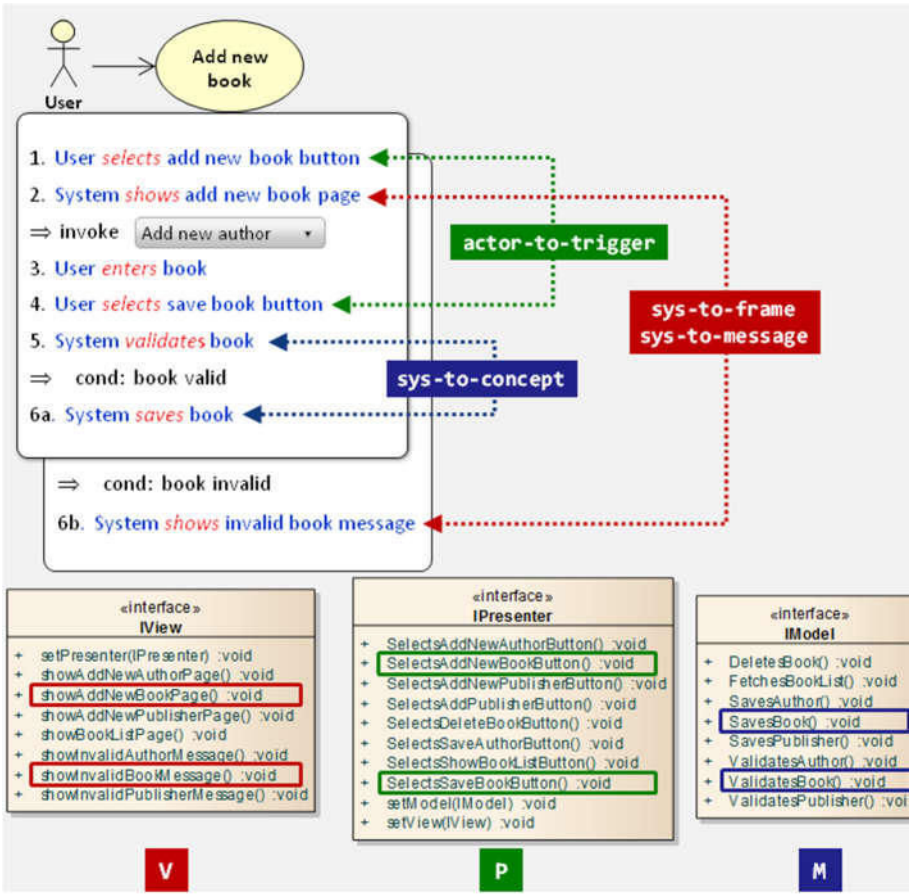

Figure 7. Generation of interface operations based on sentence types

layer (e.g. 'ShowAddNewBookPage'). Finally, System-to-Concept sentences (e.g. 'System saves book') translate into operations of the Model layer (e.g. 'SaveBook').

Figure 8 presents how scenarios can be transformed into method code of the Presenter layer. The example translates our use case in RSL into a fragment of the "PresenterImpl" class that implements the appropriate interface shown in Figure 7. Each of the scenario sentences has its representations in code marked with the sentence number. To maintain readability Figure 8 does not present classes that implement interfaces of the View and Model layers.

The first sentence is an Actor-to-Trigger kind of sentence which generates a method for handling the button event. Additionally the button is generated on the previously displayed window (in the View layer - not shown). In this particular case the use case was not invoked from any other use case so the button will be generated on the main window of the application. The action triggered by the button is based on the next sentence of the scenario (sentence 2). This next sentence is a System-to-Frame type of sentence. The generated code calls the "showAddNewBookPage" method from the View layer.

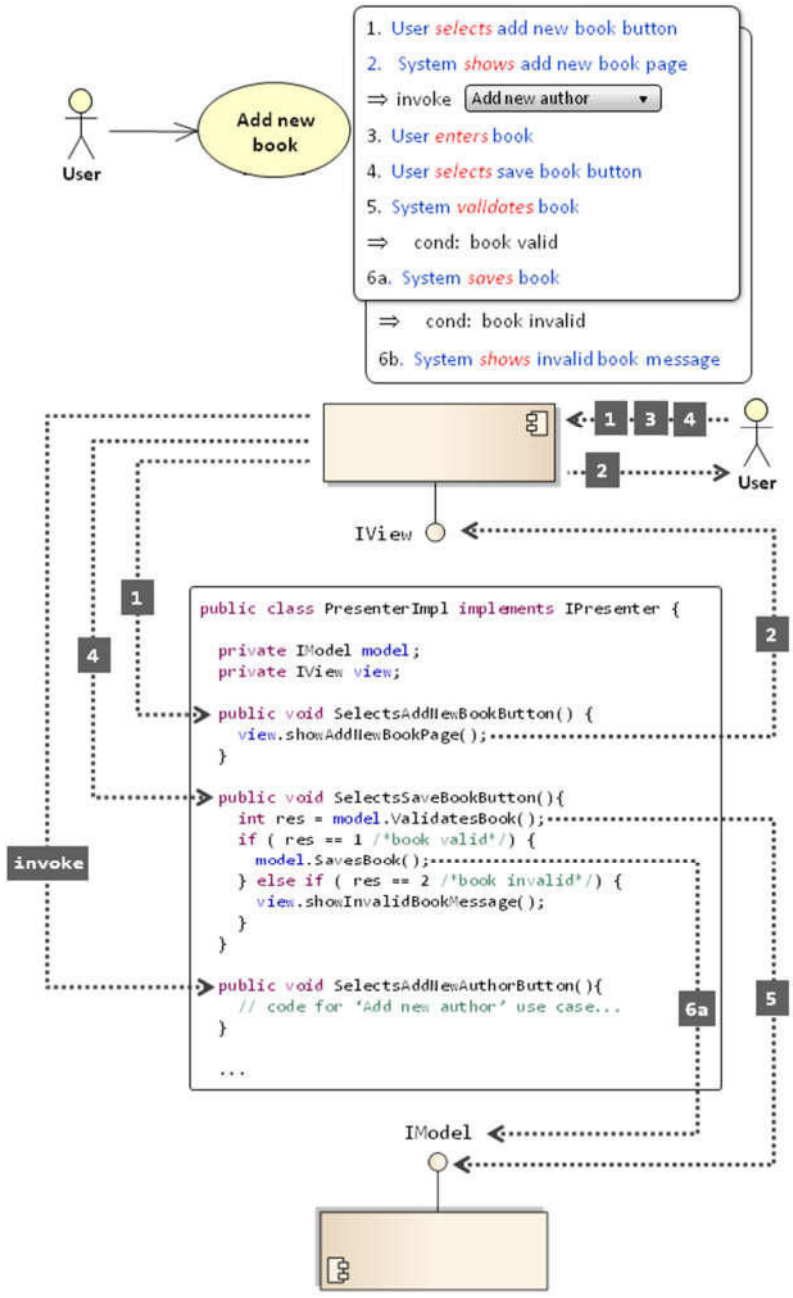

Figure 8. Generation of application logic code based on scenarios

The next sentence (unnumbered) is an “invoke” sentence which causes invocation of another use case with its own scenarios and processing sequence. Because control is at

the actor at this moment, invocation is optional. It might be triggered by the user through pressing a button which will be generated on the "add new book page". This button is determined by the first sentence of the invoked use case (not shown here). We can guess what the sentence reads, because it generated the "SelectsAddNewAuthorButton' method.

The third sentence is an example of an Actor-to-Concept sentence where the actor fills a form with data. The system does not react to this action in any way and no code is generated for the sentence. Sentence number four is an Actor-to-Trigger kind of sentence, just like sentence number one but this time the code generated for handling the button press is more complicated, because control stays on the system side. This causes generation of code from the following sentences, inside the method "SelectsSaveBookButton".

Sentence number five is a System-to-Concept type of sentence. It causes generation of a stub method "ValidatesBook"in the Model layer, which is called from the Presenter layer as the first statement within the "SelectsSaveBotton" method. The next sentence is a condition sentence. It splits scenario flow into the main and the alternative scenario. This results in generating an "if-else" structure in code. The code in the "if" block is based on the main scenario while the "else" part is derived from the alternative scenario.

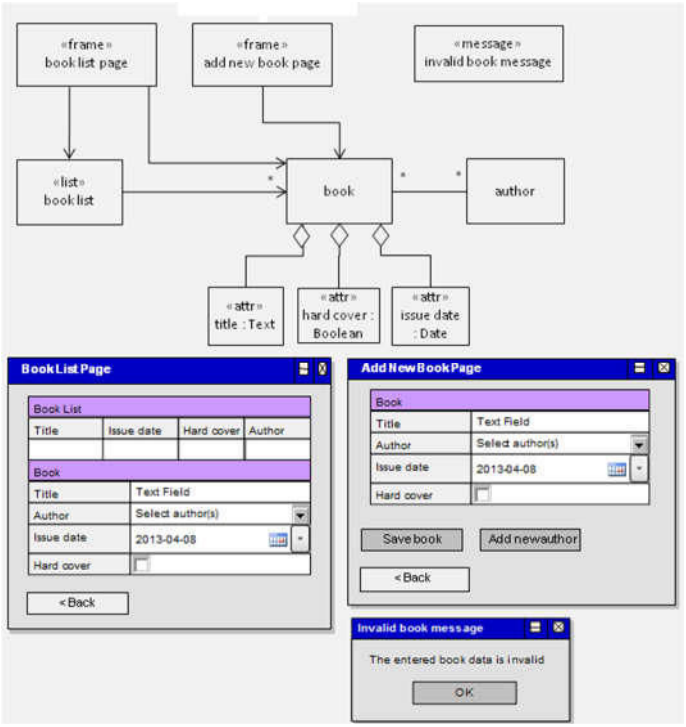

Figure 9. Generation of UI elements based on the domain model

All the objects in the above scenarios are defined as notions in the domain model presented in Figure 9. Apart from determining sentence types, the notion stereotypes can

also determine generation of UI elements in the View layer. Here we do not present this code because it is too long. Instead, Figure 9 presents the concrete look of the UI elements generated from the domain model. In this model, "Add New Book Page" is a Frame type notion which is connected with the "Book" Concept notion. The "Book" is connected with the "Author" notion and several Attribute notions. This set of relations means that "add new book page" will contain a section named after the Concept's name and filled with the Concept's attributes. Similar considerations can be done for the "BookListPage" element.

The above example shows typical code that can be obtained from RSL models. This can be easily extended to larger projects. We have conducted several case studies in various domains which were assigned to students who undertook their final undergraduate projects. This included 12 projects consisting of around 30-40 use cases. These projects have shown that the ReDSeeDS system and RSL itself are well usable by inexperienced programmers. The generated code can be easily handled and extended with the business logic that was not generated. The resulting UI is not too compelling but quite satisfactory and can be well used for rapid prototyping. What is important, the students have pointed out that they had no problems with designing the system because the transformation engine has generated clearly structured code which could be easily updated or modified.

7. Conclusion and future work

The presented approach aims to give requirements models characteristics of software programs. We introduce a Domain-Specific Language oriented towards specifying functional requirements tightly related with domain models. In this language, special emphasis is put on readability while retaining precision and introducing runtime semantics. By constructing transformation programs, combined with RSL we can produce a fully operational business application which can be directly deployed on a specific platform. The only additional effort is associated with coding of the Model layer, or - in other words - the business domain logic. This approach can be used for fast prototyping, but also final applications can be developed with success, as also validated through student projects.

Another interesting property of RSL is its clear separation of the software essence from accidental complexity. The accidental, technological complexity is encapsulated within the model transformation program. As a result, the need to develop platform-specific code is reduced to minimum. Apart from student projects, the presented solution is being validated within a large case study based on a legacy corporate banking system. Furthermore we have plans to extend experiments with university students and examine efficiency of development using the ReDSeeDS tool as compared to traditional approaches. The goal will be to undertake software development using the presented solution and in parallel – using traditional 3GL programming. This way we will be able to compare productivity and quality of the delivered systems.

Future work will also include further development of transformations to code taking into account new technologies and platforms. There is ongoing work on new transformations which will provide high-level separation of concerns and thereby high

reusability. We plan to develop a platform-independent architecture composed of several layers. The new transformations will offer several technology options to build the presentation layer, such as Google Web Toolkit, Apache Wicket, JavaFX, Adobe Flex, and technologies for mobile systems (Android, iOS etc.). The generated business logic components will consist of services that will use cloud technologies or provide services for the Internet of Things. The Data Access Layer will provide references to popular object-relational frameworks such as Hibernate, EclipseLink or pure Java Persistence API. We plan to validate these new transformations from RSL to code in a wider context of real-life projects.

8. References

[1] AndroMDA project home page. http://andromda.org/
[2] Echo Framework Home Page. http://echo.nextapp.com/
[3] Enterprise Architect Website. http://www.sparxsystems.com/products/ea/
[4] MDA website. http://omg.org/mda/
[5] Modelio Website. http://www.modelio.org/
[6] ReDSeeDS project home page. http://redseeds.eu/
[7] F.P. Brooks, No silver bullet: Essence and accidents of software engineering. IEEE Computer 20(4), 10-19, April 1987
[8] D.K. Deeptimahanti, R. Sanyal, Semi-automatic generation of uml models from natural language requirements. In: Proceedings of the 4th India Software Engineering Conference. pp. 165-174. ISEC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/1953355.1953378
[9] M. Attar, J. Miller, Agaduc: Towards a more precise presentation of functional requirement in use case mod. In: Software Engineering Research, Management and Applications, 2006. Fourth International Conference on. pp. 346-353, 2006
[10] R. Giganto, T. Smith, Derivation of classes from use cases automatically generated by a three-level sentence processing algorithm. In: Systems, 2008. ICONS 08. Third International Conference on. pp. 75-80, 2008
[11] H. Kaindl, M. Śmiałek, P. Wagner, D. Svetinovic, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, H. Schwarz, D. Bildhauer, J.P. Brogan, K.S. Mukasa, K. Wolter, T. Krebs, Requirements specification language definition. Project Deliverable D2.4.2, ReDSeeDS Project (2009), www.redseeds.eu
[12] A. Kalnins, J. Barzdins, E. Celms, Model transformation language MOLA. Lecture Notes in Computer Science 3599, 14-28 (2004), MDAFA'04
[13] S. Mustafiz, J. Kienzle, H. Vangheluwe, Model transformation of dependability-focused requirements models. In: Modeling in Software Engineering, 2009. MISE '09. ICSE Workshop on. pp. 50-55, 2009
[14] Object Management Group: Unified Modeling Language: Superstructure, version 2.2, formal/09-02-02, 2009
[15] M. Potel, Mvp: Model-view-presenter, the taligent programming model for c++ and java, 1996. URL http://www. wildcrest. com/Potel/Portfolio/mvp. pdf
[16] D. Schmidt, Guest editor's introduction: Model-driven engineering. Computer 39(2), 25-31, 2006
[17] P. Shaker, J. Atlee, S. Wang, A feature-oriented requirements modelling language. In: Requirements Engineering Conference (RE), 2012 20th IEEE International. pp. 151-160, 2012
[18] M. Śmiałek, A. Ambroziewicz, J. Bojarski, W. Nowakowski, T. Straszak, Introducing a unified requirements specification language. In: Proc. CEE-SET'2007, Software Engineering in Progress. pp. 172-183, Nakom 2007

Part II
Mobile Applications Development

Chapter 3

Assessment of Basic Architectural Decisions in Mobile Software Development

1. Introduction

Mobile software has become a rapidly developing segment of the modern software industry. The architecture of mobile software may substantially influence its quality, and an assessment of mobile software architecture can deliver important insights into the properties of software design, long before most of the development effort has been made.

There are many architecture assessment methods, though only the Architecture Tradeoff Analysis Method (ATAM) [6] has a broad record of practical application. ATAM is a general-purpose method that can be applied to virtually any piece of software. The price for that generality is the huge effort required for a full ATAM evaluation, which, according to reports by the authors of the method, amounts to as long as 70 man-days.

The idea promoted in this paper is that, in order to effectively evaluate mobile software architecture, the assessment should start from the top-level architectural decisions existing in every mobile software architecture. Then, if necessary, it should be extended by an analysis of details specific to that given mobile software. In order to achieve this, we have identified sets of:

- top-level basic decisions defining the foundations on which further development takes place, as well as,
- typical architectural tradeoffs;
- a utility tree of quality attributes;
- typical quality scenarios.

All this information can be fed into ATAM or any other general-purpose architecture evaluation method, or can be used to stimulate an informal architecture review. The above idea has been validated on an example of an ATAM assessment of a mobile banking application.

2. Related work

There is a vast amount of research on architecture evaluation, which has resulted in more than 30 methods proposed by the software architecture research community – compare e.g. [13]. However, the research record on the evaluation of mobile architecture is rather small. The following propositions seem to be most mature:

- Biel et al. [1] proposed a mobile software design method that is based on a combination of software architecture analysis and usability/user experience evaluation.
- Schäfer [5] introduces the evaluation of mobile architecture using л-calculus. However, the analysis presented in [5] is focused on the non-functional properties of examined architecture.

Bagheri et al. [4] analysed the role of various architectural concepts included in the Android platform. A variety of tradeoffs of mobile software architecture have been analysed by Barnett et al. [8]. A mobile software quality model has been drawn up by Franke and Kowalewski [11]. According to those researchers, the various quality attributes such as usability, flexibility, portability, extensibility etc. depend on each other.

3. Specific nature of mobile software

Mobile software is supposed to run on mobile devices like smartphones or tablets. The mobile application architect has to take into account several issues specific to this type of software, namely, the mobility of the user and the device on which the application is used, as well as the unreliability of network, the limited capacity of the user interface, restrictions on power consumption as well as limited CPU and memory resources. From a functional perspective, mobile applications intensively use mobile device sensors (accelerometers, gyroscopes, GPS, etc.). It should be underlined that the security and diversity of the base software – different operating systems and their versions [2], [3], are also important aspects of mobile applications.

3.1. Reference architecture

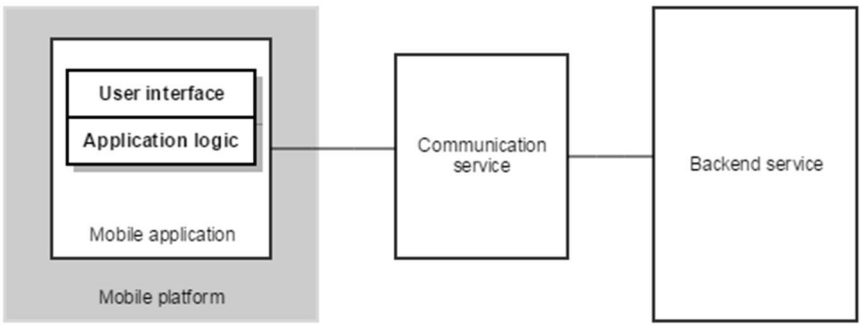

Figure 1. Reference architecture elements of mobile software

Mobile software consists of a mobile application that offers application services to its end user and exposes its logic through the user interface. Most applications are not stand-alone, but demand some kind of backend service that delivers information from the application ecosystem. The communication between the application and its backend is

provided by communication service. Communication service is responsible for using communication protocols and ensuring the security of the communication channel.

4. Information pack for mobile software evaluation

Below, we present the set of information that is supposed to facilitate the assessment of mobile software architectures.

4.1. Fundamental decisions in mobile software architecting

Architecting a mobile application, regardless of its functionality, requires making the following fundamental architectural decisions:

- Choice of application style: web application, native application (i.e. dedicated to a given mobile platform, such as iOS or Android) and multiplatform application;
- Communication service implementation: SOAP Web Service, RESTful, other,
- Communication pattern: synchronous vs asynchronous,
- User interface pattern: nested doll, tabbed view, page hierarchy, hub and spoke or dashboard,
- Backend service deployment: cloud (cloud provider) or on premises,
- Functionality responsibility split between the mobile application and backend service – centralised or distributed,
- Selection of mobile platform: Android, iOS, Microsoft, etc.

4.2. Mobile architecture stakeholders and their concerns

The typical stakeholders of mobile software and their concerns are listed in table 1.

Table 1. Mobile architecture stakeholders

Stakeholder	Concerns
architect	functional and non-functional requirements
owner/sponsor	business value
	costs of development and maintenance
	modifiability
developer	functional requirements
	modifiability
user	usability
	performance
administrator	maintainability
	performance

4.3. Utility tree for mobile software

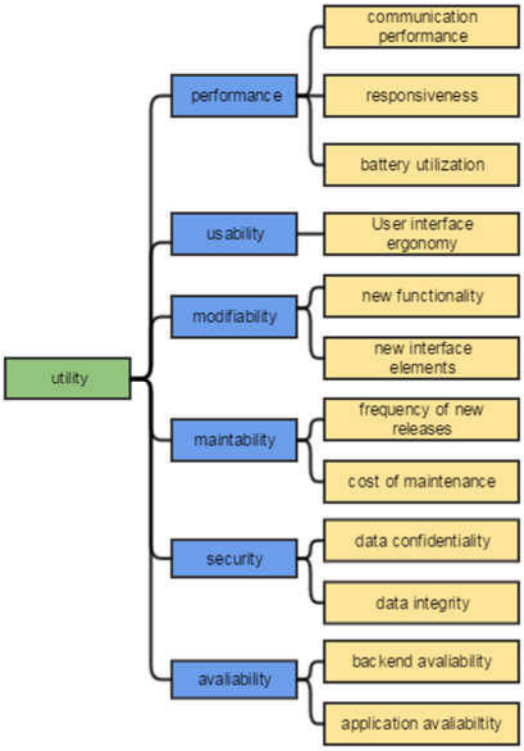

Figure 2. First 3 levels of a typical utility tree for mobile software

Figure 2 presents a utility tree for mobile applications. The scenarios are specific for mobile software applications, so they have to be completed by the architect of the software. However, there are some common scenarios for almost every application. Examples of these scenarios are listed below.

4.4. Typical quality scenarios

ATAM defines three types of scenarios: use case scenarios, growth scenarios, and exploratory scenarios. These are discussed below in the context of mobile software.

4.4.1. Use case scenarios

Use case scenarios cover the standard usage of a designed software system.

Table 2. Typical use case scenarios for mobile software architecture

Quality attribute	Scenario
Performance	retrieve data from the backend service and present it to the user in a time not longer than N s.
Performance	the mobile application will use less than N % of battery in a time not longer than T s.
Usability	mobile application change of view in the user interface not longer than T s.
Availability	the mobile application will renew the data connection after network reconnection in less than T s.
Availability	the application will detect network unavailability and switch to offline mode
Security	communication to the backend service is properly secured against data leaks or unauthorised modification
Security	the mobile application does not allow access to data without authorisation
Maintainability	a new release of mobile application not less than every N weeks
Maintainability	a critical error fix in less than N hours

4.4.2. Growth scenarios

Growth scenarios represent typical future changes to a system.

Table 3. Typical growth scenarios for mobile software architecture

Quality attribute	Scenario
Modifiability	add a new view in the user interface in less than N person-day effort
Modifiability	add new mobile platform support in less than N person-day effort
Modifiability	add a new functionality in less than N days
Modifiability	rebuild the user interface (refresh) in less than N days

4.4.3. Exploratory scenarios.

Exploratory scenarios expose the limits or boundary conditions of the current design.

Table 4. Typical exploratory scenarios for mobile software architecture

Quality attribute	Scenario
Modifiability	change of the communication service implementation from REST to SOAP, or the other way around, in less than N days
Modifiability	change of the implementation of the backend service, for example changing a database from SQL to NOSQL in less than N days
Modifiability	add a new data source from a new partner of the mobile software in less than N days
Availability	change of location of the backend service, for example moving from on-premises to cloud, or changing between clouds

4.5. Basic tradeoff points of mobile software architecture

A list of basic tradeoff points has been presented below:

- **Selection of a user interface architecture** is a tradeoff point because it is important for both performance and usability. An interface that is highly ergonomic and user friendly can be also resource-consuming and provide insufficient performance.
- **Choice of application style**: there are three main application possibilities: web application, native application and multiplatform application. The right choice of application is crucial for three quality attributes: performance, usability and maintainability. Web applications are weak from performance and usability attribute points of view, but strong in maintainability. Native applications are best in performance and usability, but maintainability is poor. A multiplatform approach has a positive impact on maintainability, but is negative for performance. The impact on the usability attribute depends on other decisions.
- **Communications service implementation** (REST vs SOAP) is a tradeoff point important for performance, maintainability and modifiability. This decision is dependent on the backend server architecture. Implementing REST in a communication service supporting performance is negative for maintainability and modifiability, whereas a backend server supports only SOAP communication.
- **Communication pattern** (synchronous vs asynchronous) is a tradeoff point that has an influence on security, performance and availability. An asynchronous pattern provides better performance and is less vulnerable to temporal communication deterioration, but is less secure than a synchronous pattern.
- **Backend service deployment (cloud vs on-premises)** is a tradeoff point between availability, maintainability and security. In order to make this multidimensional decision, an architect has to take into account different conditions that have an influence on quality attributes:
  - *total cost of ownership* (TCO) of on-premises deployment vs cloud deployment,
  - legal and regulatory compliance of the cloud provider,
  - check whether cloud deployment is in line with the sponsor (owner of mobile software) IT strategy,
  - check whether the cloud provider is reliable.

Figure 3. presents a diagram of the relationship between identified tradeoff points and quality attributes of a utility tree of mobile software.

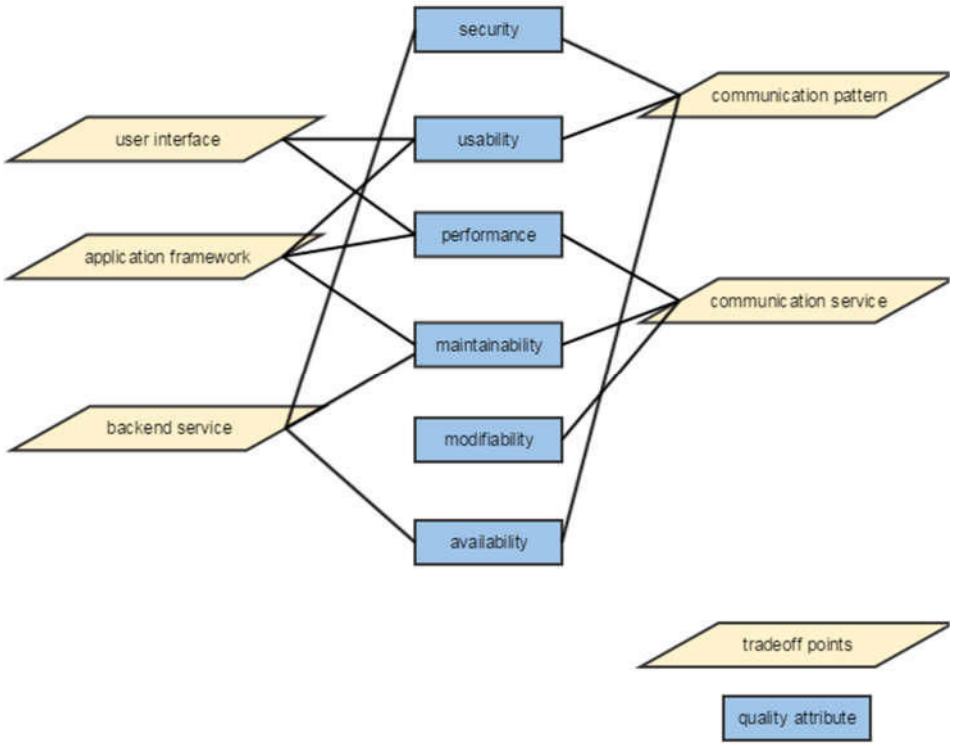

Figure 3. Tradeoff points impact on quality attributes

5. Application example

As a practical validation of using the information pack for mobile applications, an assessment has been made on the application *BANKApp,* which is an internet banking application for *BANK Enterprise*. The user of BANKApp can view the balance of his bank accounts and display transaction history. The user can also change the limits of ATM withdrawal, debit cards and bank transfers limits. In order to decrease security requirements, BANKApp does not allow payments or bank transfers to be performed (business-level decision).

5.1. BANKApp business drivers

The business drivers collected for BANKApp are as follows:

- Business goals and business context:
  - to provide bank customers with access to their accounts on mobile devices,
  - to retain current customers and acquire new customers,

  - to follow trends in internet banking,
- Key functions of the mobile application:
  - to display data fetched from the server,
  - to maintain connection with the backend service,
- Known technical, organisational, economical limitations:
  - technical limitations of mobile devices: limited battery, CPU, memory, low screen size,
  - limited budget and time,
  - should be available on mobile devices based on: Android, iOS and Windows Phone,
  - application distributed via app stores,
- Main stakeholders:
  - the business owner – BANK Enterprise,
  - the BANK customers,
  - the system administrator,
  - the system architect,
  - the development team,
- Architectural drivers
  - system availability 99% – high availability, though some unavailability is possible,
  - system security 99.9999% – security at a very high level, even an insignificant security incident can cause a bad reputation and loss of customers,
  - easy modifiability in order to follow the trends on the internet banking market,
  - performance 99% on at least 80% of devices running on Android, Windows Phone and iOS platforms,
  - maintainability has to be economically justified.

5.2. BANKApp utility tree

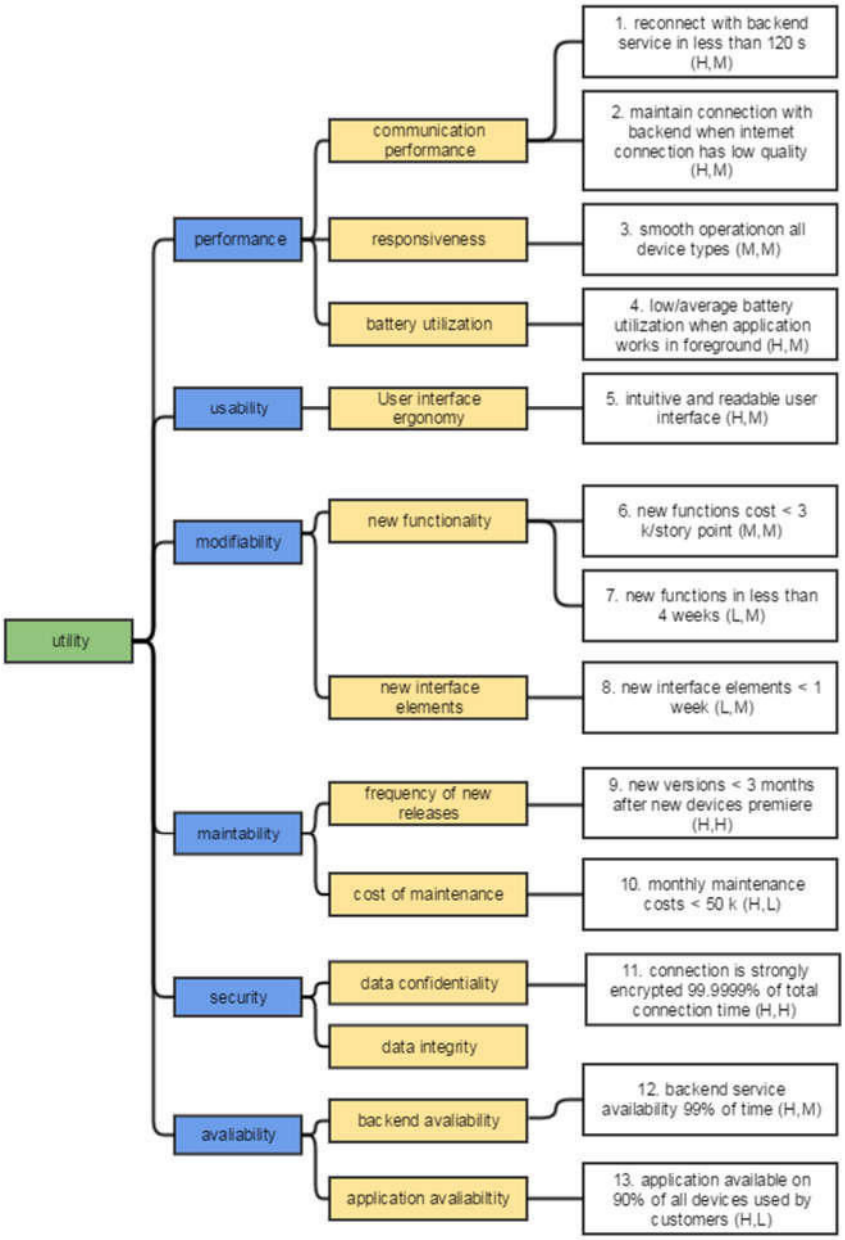

Figure 4. Complete utility tree for BANKApp

5.3. BANKApp architecture

The architecture of BANKApp follows the reference architecture presented in figure 1. The mobile application contains a user interface with application logic and security modules. The application is installed on a mobile device and communicates with the bank through a REST API gateway that enables information retrieval from bank transaction system. The implementation of communication service has been fixed (as a RESTful API) by already existing backend system; as a result, this decision is not a tradeoff point in this case.

5.3.1. Discussing architectural decisions and tradeoffs

According to figure 3, the key decisions that shape the architecture of BANKApp are tradeoff points for mobile applications. The mobile software architect is responsible for facilitating discussion with stakeholders on their preferences concerning quality attributes. Each decision has an impact on two or more quality attributes. While evaluating this impact utility tree, the scenario and quality attribute measures have to be taken into account. With regard to BANKApp, the following discussions with stakeholders on tradeoffs were registered:

- Communication service – **REST**

  Communication service is not a subject to architectural decision, because there is already an existing backend service that leads to a RESTFul communication service.

- Communication pattern – **synchronous**

  Communication pattern is a tradeoff important for security and performance that should be considered by taking into account scenarios 1, 2, 3 and 11. Synchronous communication is the better choice than asynchronous one, as we can see from the scenarios mentioned above. A synchronous communication pattern is better secured, which is crucial for BANKApp. Despite the fact that it has a negative influence from performance perspective, tradeoff security wins.

- User interface – **hub and spoke and filtered view**

  The decision on the user interface can be validated on the basis of scenarios 1-4, as this decision is a tradeoff point for performance and usability. For the reason that there are more alternatives than a simple choice between two or three options, the architect should firstly propose a mock-up of the user interface presented in figures 5, 6 and 7. Secondly, this mock-up has to be evaluated against scenarios 1-4. Finally, evaluation outcome is positive when the interface is approved by stakeholders as intuitive, and no performance issues were found.

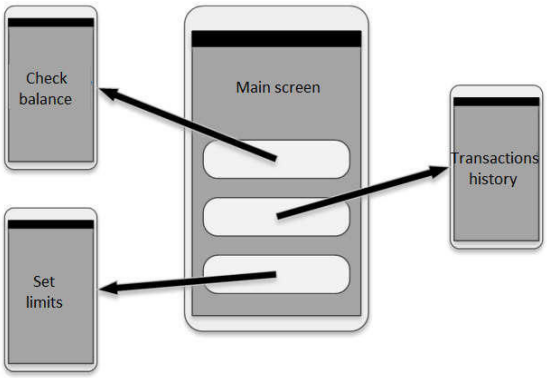

Figure 5. Main screen as hub and spoke interface

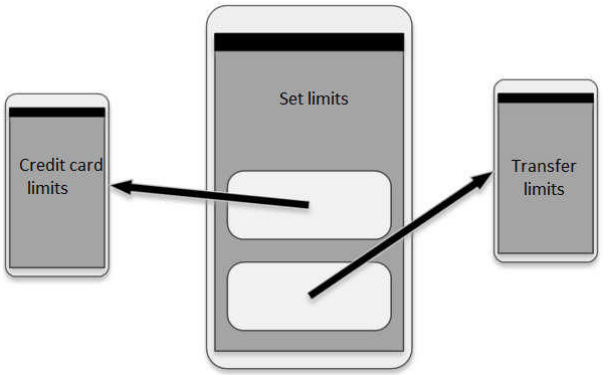

Figure 6. Set limits as hub and spoke interface

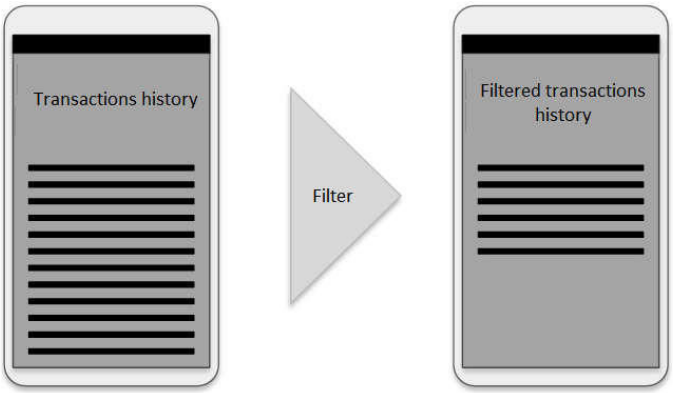

Figure 7. Transactions history as filtered view interface

- Application style – **native**

  The decision on the application framework is a tradeoff point regarding scenarios 1-5, 9 and 10. The stakeholders decided that a native application will be applicable for performance and usability, but will not hinder the maintainability scenarios.

- Backend service – **on premises**

  The backend service is already implemented, but its deployment is still an architectural decision. The stakeholders decided that the backend service should be deployed on premises for security reasons. This alternative ensures full control over the security of the system, and does not lead to compliance issues. The bank's systems are subject to many legal regulations concerning, in particular, customer data protection and confidentiality. The lack of third party (cloud provider) ensures compliance with those regulations. This decision has a negative influence on maintainability and availability, but the stakeholders are fully aware of its implications.

6. Summary

The idea of focusing the analysis on a set of architecturally significant decisions can be both related to the concept of minimalist architecture promoted by Malan and Bredemeyer in [7], as well as that inherent in ATAM [6, 13]. The main contribution of this paper is the proposal of an information pack that contains information useful for an evaluation of the foundations of mobile software. This goes beyond the exemplary issues and tradeoffs identified in [8].

The paper promotes the idea of facilitating mobile architecture evaluation with a predefined information pack comprising fundamental architectural decisions, typical tradeoffs, stakeholders and concerns and quality scenarios. All this can be fed into the architecture evaluation process in order to speed up and facilitate the evaluation. The experiments carried out have shown that this actually works in practice. Further research may lead to the proposition of a method specially tailored for the needs of the assessing mobile software architectures.

References

[1] B. Biel, T. Grill, V. Gruhn, Exploring the benefits of the combination of a software architecture analysis and a usability evaluation of a mobile application. Journal of Systems and Software, 83(11), 2031-2044, 2010

[2] M.E. Joorabchi, A. Mesbah, P. Kruchten, Real challenges in mobile app development. In Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on, pp. 15-24, IEEE, 2013

[3] J. Dehlinger, J. Dixon, Mobile application software engineering: Challenges and research directions. In Workshop on Mobile Software Engineering, Vol. 2, pp. 29-32, 2011

[4] H. Bagheri, J. Garcia, A. Sadeghi, S. Malek, N. Medvidovic, Software architectural principles in contemporary mobile software: from conception to practice. Journal of Systems and Software, 119, 31-44, 2016

[5] C. Schäfer, Modeling and analyzing mobile software architectures. In European Workshop on Software Architecture, Springer Berlin Heidelberg, pp. 175-188, September, 2006
[6] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, J. Carriere, The architecture tradeoff analysis method. In Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on, pp. 68-78, 1998
[7] R. Malan, D. Bredemeyer, Less is more with minimalist architecture. IT professional, 4(5), 48-47, 2002
[8] S. Barnett, R. Vasa, A. Tang, A conceptual model for architecting mobile applications. In Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on, pp. 105-114, IEEE, 2015
[9] V. Rahimian, R. Ramsin, Designing an agile methodology for mobile software development: A hybrid method engineering approach. In Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on, pp. 337-342, IEEE, 2008
[10] S. Pastore, Approaches and methodologies for mobile software engineering. Advances in Computer Science: an International Journal, 4(5), 14-22, 2015
[11] D. Franke, S. Kowalewski, C. Weise, A mobile software quality model. In Quality Software (QSIC), 2012 12th International Conference on, pp. 154-157, IEEE, 2012
[12] B. Len, C. Paul, K. Rick, Software architecture in practice. Boston, Massachusetts Addison, 2003
[13] A. Zalewski, Modelling and Evaluation of Software Architectures. Warsaw University of Technology Publishing Office. Warsaw, 2013

Chapter 4

Emotional Intelligence in Mobile Games

1. Introduction

Non Player Characters (NPCs) have a very important role to play in video games. They may act as allies, helping to achieve the main goal, or as competitors, making every step harder during the gameplay. Regardless of the role they play, virtual agents are expected to be believable characters, behaving in a way that allows the player to step deeper into the lore of the game. Usually, the character's performance is modeled by scripting automatic behavior, triggered by certain circumstances induced during the gameplay. However, this method produces a gameplay that is repetitive, and thus unnatural. The goal is to create more complex, human-like characters that can contribute to a more unpredictable and involving gaming experience. One of the most vital aspects of every character in films, books and video games, including both human beings and animals, are emotions. It is the ability to express emotions that makes the virtual character believable and natural. This, in turn, helps to create a bond between the user and the virtual world. Thus, a great deal of attention is paid to the modeling of the NPC emotions component, as a crucial element for the game's success. This marks a shift towards a greater focus on the interpersonal aspects of the gameplay, aiming to increase the enjoyment of players. Nevertheless, virtual agents that are emotional, autonomous and driven by certain motivations are still rather rare in currently available video games.

2. Cognitive Psychology models of emotions

The quantitative analysis of human emotions is a broad area of psychological research. There are many various approaches that contribute to creating a synthetic emotional framework to simulate complex interactions driven by emotional motivations of game agents. The major theoretical approaches to the simulation of emotion include: Appraisal Theory, Dimensional Theory, as well as Hybrid Theories, which combine the most advantageous aspects of both aforementioned approaches.

2.1. Appraisal Theory

Appraisal Theory is the most widely applied approach to emotional models. It suggests that emotion is based on patterns of individual judgment that arises from the connection between the events and the individual's thoughts, intensions and desires, commonly known as the person-environment relationship. These individual evaluations,

formalized through reference to devices, such as appraisal variables, describe aspects of the personal relevance of events [1].

Therefore, the Appraisal Theory sets appraisal as the main process involved in the modeling of emotions. Emotions may be considered as discrete elements or as configurations of appraisals. There is a variety of appraisal models, such as Roseman's model, Scherer's model, or the OCC model proposed by Ortony, Clore and Collins. The OCC model is the most widely applied model of emotion, which gave birth to many architectures implementing the appraisal theory.

2.2. The OCC model

The OCC model provided the foundation for many modern approaches to artificial emotions. Its main assumption is that emotions are reactions to the attributes of objects, events and actions of the character itself and other characters. During the appraisal process, objects, actions and events are evaluated against a set of predefined criteria and, as a result, emotions with varying intensities are generated [2].

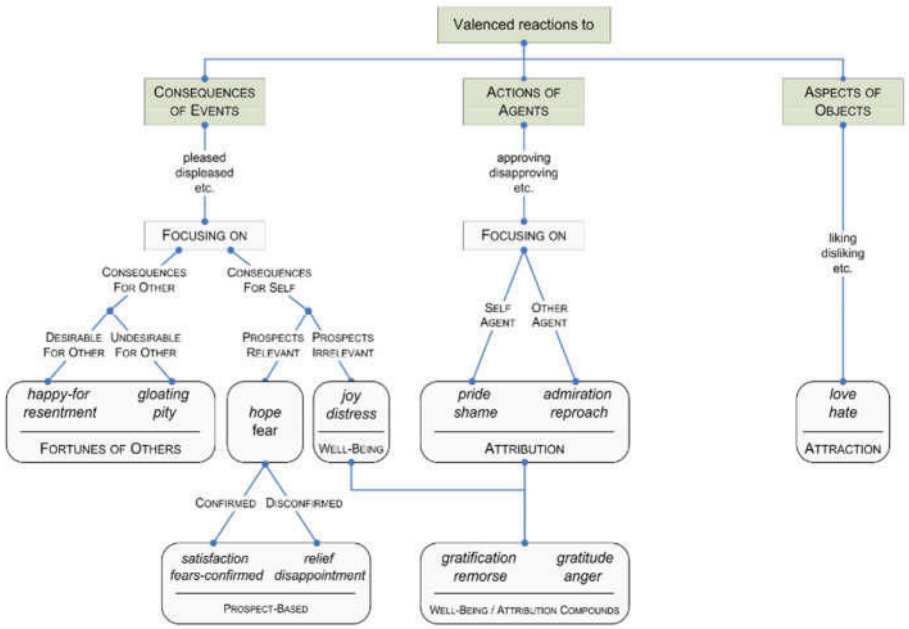

Figure 1. The OCC model [4]

The process of generating emotions in the OCC model is shown in Figure 1. The agent's appraisal (evaluation) of an object, based on certain preferences or attitudes that the agent has towards that object, leads to a specific reaction and decision how to act. The final result of the appraising process is either love or hate. The consequences of events that already have happened are appraised through an analysis of their impact on the agent's goals. This determines if the agent is willing to perform an action or be part of an

event. Joy and distress are effects of desirable and undesirable events considered as consequences involving the agent himself [4].

However, the appraisal process may be affected by events that carry consequences for other agents or events that are consequences of actions performed by other agents. As a result, social emotions like pity, admiration or reproach are generated. Moreover, a noticeable phenomenon is the ability to feel pride or shame of someone else's actions. A good example are parents being proud of their child. This phenomenon is connected with the agent's ability to identify with the other agents, which may involve common objectives or evaluation of emotions towards each other [4].

This leads to the crucial question of how to find the right balance between the agent's empathy and their need to achieve a specific goal. This problem can be solved by defining abstract and general goals. Such an approach provides the ability to reduce significantly the number of predetermined goals. The appraisal process involves expectations which may have a positive or a negative desirability value for the agent, generating such emotions as fear or hope that something will or will not happen. The aftereffects of those expectations are the feelings of disappointment or relief, which are likely to influence future decisions of the agent. A practical example of an architecture for emotional agents is FAtiMA [1], [4].

2.3. FAtiMA architecture

FAtiMA (Fear not Affective Mind Architecture) is an Agent Architecture with planning capabilities designed to use emotions and personality traits to influence the agent's performance. It consists of modular components which may be combined to create various appraisal process scenarios [5].

FAtiMA Modular is composed of a Core layer, on which components can be added to extend it with new functionalities. The Core is a template defining how the Agent Architecture works. Figure 2 provides the visualization of the FAtiMA core with its basic functionalities [5].

An agent is able to gather perceptions from the environment which are then used to update its memory and begin the appraisal process. The result is kept in the affective state (emotions and moods) and, in the further stage, influences the agent's action selection process. The appraisal process is divided into 2 separate steps: appraisal derivation and affect derivation. The former is used to evaluate the relevance of the event to the virtual agent and determine the appraisal variables. The latter takes the appraisal variables as input and generates the resulting states. Those states are emotions or moods, according to the Appraisal Theory [5].

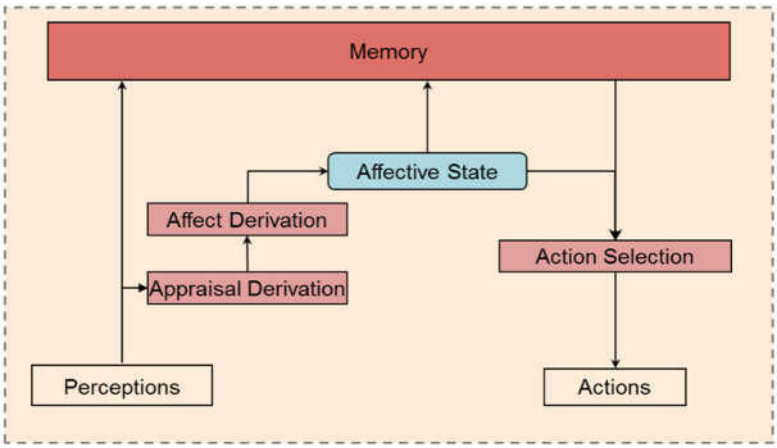

Figure 2. FAtiMA core architecture [5]

It is vital to underline the fact that the core itself will not provide any results without the other components. Specific implementations are provided by adding particular components, which take input data and generate the resulting affective states that are processed to generate the agent's actions. However, it is not required for the architecture to implement all functionalities [5].

An Appraisal Frame structure is designed to store partial results of the appraisal variables, generated during the appraisal process. It is also used to allow any appraisal derivation module to access those pieces of information. This structure is usually connected with the event. Moreover, it must store the contribution of all modules to an appraisal variable and use a set of rules to establish its final value. Such an approach provides the ability to maintain a multi-layer engine architecture based on the Core and the functionality components. Figure 3 shows an example of an Appraisal Process according to the multi-layer concept of FAtiMA [5].

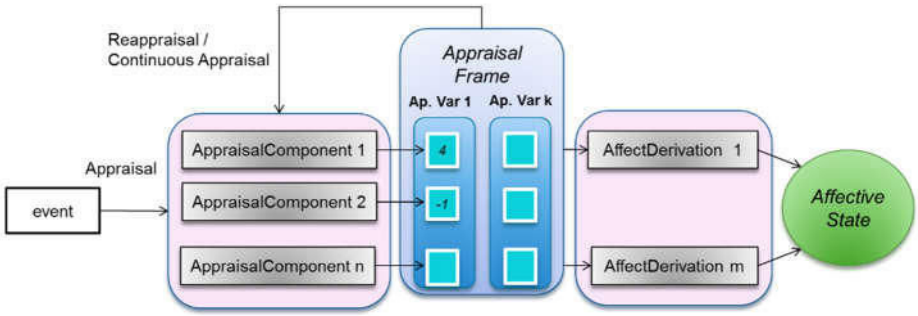

Figure 3. Appraisal Process [5]

When the agent perceives a new event, an Appraisal Frame is created to initialize the appraisal process. The arrow placed over the appraisal frame represents a loop indicating that appraisal is a cyclical process. The amount of cycles depends on the number of components and the appraisal itself in every cycle. In the presented example, the first

component defines the variable with value 4, which means a positive result. However, during the next steps of the appraisal process, another component changes the value of the variable to slightly negative (-1). Every redefinition in the appraisal frame, has to be reflected in the emotional state of the agent. This is handled by Affect Derivation components, the main task of which is to update and generate the agent's emotions. The resulting data are used as a basis for selecting the dominant emotion and its intensity. This decision may be also affected by other modulation factors, such as personality or cultural bias. It is important to note that all derivation components are independent, which means that the agent can feel several emotions at the same time. The intensity determines the final emotional state [5].The modular architecture of FAtiMA enables the use of many different interchangeable components to execute the appraisal process. As already mentioned, the Core alone will not provide the expected results. There are many possible emotion processing modules that can be applied to ensure that the agent reacts in a realistic fashion. For example:

Reactive Component - this component evaluates the value of the appraisal variables by predefined emotional reaction rules. Whenever an event takes place, the reactive appraisal matches the event against a set of emotional rules. A rule can specify a particular value for every appraisal variable and connect it with a certain event.

Deliberative Component - this component is responsible for the ability to plan and achieve the goals of the agent. It also uses the states of plans recorded in memory to generate appraisal variables, referred to as goal status, goal conduciveness and goal success probability.

Motivational Component - this component generates primary human drives, such as energy and integrity. It provides the ability to choose between competing objectives in the deliberative component. It defines the relation between a need and the utility of a goal that contributes positively to fulfilling that need. Moreover, it determines an event's desirability on the basis of an influence it had on the agent's drives.

Cultural Component – this component handles the cultural-dependent behavior of agents by the use of rituals, symbols and cultural aspects. It automatically sets the praiseworthiness appraisal variable determined by cultural values. For example, if the agent is a part of a collectivistic culture, a praiseworthy action is one that positively influences the needs of other agents to the detriment of the agent's own needs [5].

These are the basic FAtiMA components, however, it is possible to use many other components in different combinations, allowing the architecture to adapt to different scenarios. Table 1 shows an association of emotions types with appraisal variables and their sets.

Table 1. Assocation between occ appraisalvariables and occ emotion types [5]

Appraisal Variables	***Associated Emotion Types***
Desirability	Joy, Distress
Desirability for others, Desirability	Happy for, Gloating, Pitty, Resentment
Praiseworthiness	Pride, Admiration, Shame, Reproach
Praiseworthiness, Desirability	Gratification, Gratitude, Remorse, Anger
like	Love, Hate
Goal status, Goal conduciveness, Goal sucess probability	Hope, Fear, Relief, Satisfaction, Disappointment

3. Dimensional Theory

According to dimensional theories of emotion, all affective phenomena are understood as points in a continuous dimensional space. These theories reduce the emphasis on the categorization of the emotion, focusing rather on the concepts of mood, affect or core affect. Thus, an entity may be in just one affective state at a given moment in time. The space of possible core affective states is defined in terms of broad, continuous dimensions. The most popular representation of the dimensions theory is the PAD three-dimensional model, proposed by Mehrabian and Russel, in which these three dimensions correspond to Pleasure, Arousal and Dominance [1].

The Core affect is usually represented as a continuous time-varying process which is shown at a given period of time by a point in a 3D space. That point can be moved around by eliciting events. In computational dimensional models, there are detailed mechanisms that explain the changing process of this point. It is analyzed how the core affect decays to some resting state or incorporates the impact of dispositional tendencies like personality or temperament [1].

3.1. PAD Dimensional Emotional & Temperament Models

The works on the use of the three dimensions for classifying, measuring and applying emotions propose a framework for representation of emotional states and temperament of an agent. Therefore, the space defined by dimensions can also represent longer-lasting emotional states, defined as moods [8].

The PAD model presents three orthogonal scales of emotions: pleasure-displeasure (P element, representing affective states), arousal-nonarousal (A element, mental and/or physical activity) and dominance-submissiveness (D element, for instance, the control over current events) [8].

For example, eating is considered as arousing and pleasant. These emotional affects vary depending on the amount, variety, presentation and quality of the food that is consumed. Physical activity (jogging, football, gym, swimming, etc.) reduces arousal and greatly increases pleasure and dominance. Moreover, it can affect anxiety and depression moods by reducing those states [8].

Every emotion state known from the OCC model can be easily described using the dimensional PAD model. Mapping is based on finding the influence on every dimension of an emotional state, for instance:

- Anger: (-0.51. 0.59, 0.25)
- Fear: (-0,64, 0,6, -0,43)

Every of those values represents a point in the PAD space, in the order illustrated above [1], [8].

Therefore, the PAD model provides the ability to describe the character's temperament, which is defined as an average of the character's emotional states across a representative variety of life situations. It is also based on a variety of stable or lasting emotional characteristics like emotional traits or emotional predispositions. An example of the architecture using the PAD model is WASABI.

3.2. WASABI Architecture

The WASABI architecture is built of bodily emotion dynamics with cognitive appraisal. It is used to simulate infant-like primary emotions and cognitively elaborated secondary emotions. Its specification is based on the concept of Emotions and Moods [7].

- Emotions are considered as current states with a specific quality and intensity. Those states can be classified into primary and secondary ones. Note that every emotion has a positive or a negative value and, in comparison to mood, it is a short-lived state. Secondary emotions differ from primary emotions in that they are:
  - built of more complex data structures,
  - more dependent of the memory and reasoning processes of the agent,
  - responsible for the expression and verbalization with respect to other embodied interaction agents.
- Mood is understood as a background state with a much flatter affective quality than emotions. This state is not derived from the PAD space, but built as an agent's overall feeling of well-being on the bipolar scale of valance just before the mapping into the PAD space is achieved. Non-neutral moods are always slowly set back to a neutral state of mood. However, this process is much longer lasting than emotional valence [7].

The primary emotions are inborn affective states triggered by responses to potentially harmful events. A good example are involuntary reflex actions in response to stimuli. Those emotions are represented in the PAD space as points. WASABI defines nine primary emotions: angry, annoyed, bored, concentrated, depressed, fearful, happy, sad and surprised [7].

The elicitation of secondary emotions involves a "thought process", in which stimuli are evaluated against the previously gained experience and generated expectations. All emotions involved in the OCC model that are used in the architecture are classified as secondary emotions, since they are considered as conclusions, which are: fear, hope, relief, disappointment, satisfaction and fears-confirmed[7].

Every stimulus perceived is appraised parallelly by conscious and non-conscious processes, which leads to the development of an "emotional impulse". This results in "emotional dynamics", which is an element of the agent's virtual embodiment and from which mood, pleasure and arousal are fluently derived. At this point, the PAD space is used. It directly elicits primary emotions with a certain intensity. The PAD space is also used as an "awareness filter", ensuring mood-congruency of primary and secondary emotions. The results of "aware emotions" are assembled in the final reappraising process before the implementation of the action selection process. Figure 4 presents the visualization of emotion processing in the WASABI architecture [1].

The cognitive layer of the WASABI architecture makes it possible to update the memory of the agent and generate expectations. In the evaluation process, it enables the generation of secondary emotions [1].

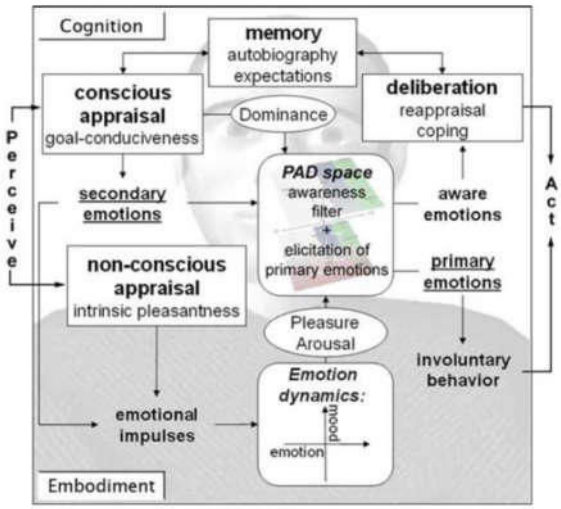

Figure 4. WASABI Architecture [1]

4. ALMA Model as Hybrid architecture

The ALMA model represents a hybrid approach based on the OCC Model and the Dimensional Theory. It is split into three layers based on time durability of components influencing the agent's actions [6]:

- long-term layer, defined by the personality,
- middle-term layer, defined by the moods and
- short-term layer, defined by emotions

This division is presented in Figure 5:

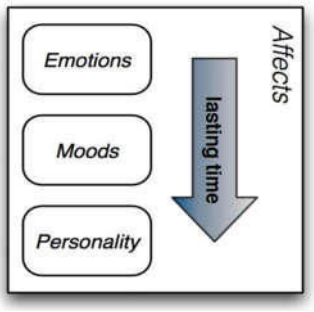

Figure 5. ALMA layer model [1]

The time arrow points towards the most durable affecting components. The long-term layer defined by the personality is basically centered on five factors and their constituent traits:

- Openness: appreciation for art, emotion, adventure, unusual ideas, curiosity and variety of experience,
- Conscientiousness: a tendency to present self-discipline stance, act dutifully and aim for achievement; planned actions, not under influence of the moment.
- Extraversion: energy, positive emotions, urgency and the tendency to seek company of others
- Agreeableness: a tendency to present compassionate and cooperative stance,
- Neuroticism a tendency to feel negative emotions with ease like anger, anxiety, depression, or vulnerability [6].

The personality is the initial stance over the mood of the character, usually set in a "neutral" state. Since it is not easy to influence the character's personality, it is considered as constant [6].

The second layer of the ALMA model solves the mood aspect of the character. Moods are stable states having high influence on the cognitive functions represented by the PAD model [6].

The last layer of the ALMA model is based on the emotions generated by an event perceived by the agent. The emotions are short-term affects connected to a certain event, action or object. They usually quickly fade away. All events perceived are categorized according to: (1) the influence that they have on the object, origin and observer of the action (2) the action itself, and (3) the objects involved in the action and the perception and relevance that it has to the observer [1], [6].

With every appraisal of an action or event, the ALMA model generates active emotions that decay over time. Those emotions are used as the input of the pull and push mood change function that is presented in Figure 6.

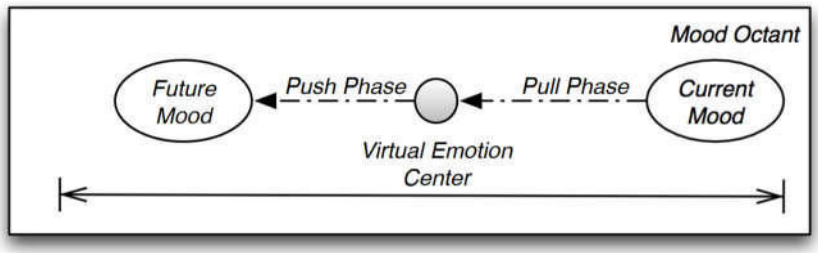

Figure 6. ALMA Pull-Push Mood Change Function [1]

If the current mood position is between the PAD space zero point and the virtual emotion center, then the current mood is pulled to the virtual emotion center in the pull phase. If the current mood is over the virtual emotion center, the current mood is pushed away in the push phase. This mechanism reflects the concept that an individual's mood intensifies as the number of experiences contributing to that mood increases[1], [6].

5. Conclusion

The integration of emotional intelligence into virtual agents has become one of the central issues of video game development. The simulation of emotions for NPCs includes such processes as generation of emotions in response to certain stimuli and modeling the effects of these emotions (decision-making engines). All these elements are crucial for an agent's credibility. The present study has examined the advantages and limitations of the major approaches to the simulation of emotions for virtual agents. The Appraisal Theory quantifies emotions into strictly defined states, such as anger or happiness. The sharp division between these emotion categories results in the agent's unnatural ("mechanical") behavior. Such an approach precludes the possibility to take into account such aspects of human psychology as mood.

A possible solution to this problem is provided by the Dimensional Theory approach, where all emotions are considered as points in the three-dimensional PAD space. In this way, it is possible to generate a more "fluent" agent behavior, which incorporates many other affective characteristics, such as mood or personality traits that may influence the agent's emotional state. Mutual interference between emotions is also possible. However, a major drawback of this approach is the difficulty in defining certain emotional states reached by the agent.

Hybrid solutions, integrating the advantages of both aforementioned theoretical approaches, define the appraisal engine based on memory and experience using the PAD space, where the emotions are visualized as points. Thus, it is much easier to define if a given emotional state falls within a particular category, such as joy or anger. Such an approach also ensures a smoother transition between emotional states, while also making them easy to categorize. However, it is still possible to allow mutual interference between emotions and incorporate longer-lasting aspects like moods. Thus, the greatest strength of the hybrid approach lies in its flexibility.

References

[1] L. Pena, Prof. Dr. S. Ossowski, Prof. Dr. J. M. Pena, "Gaming with Emotions: An Architecture for the Development of Mood-Driven characters in Video Games," Madrit, 2013.

[2] C. Bartneck, "Integrating the OCC Model of Emotions in Embodied Characters: Applications, Methods and Reasearch Challenges," 2002

[3] C. Bartneck, M. J. Lyons, M. Saerbeck, "The Relationship Between Emotion Models and Artificial Intelligence," in SAB2008: Workshop on The Role of Emotion in Adaptive Behaviour and Cognitive Robotics, Osaka, 2008

[4] H. Kessler, A. Festini, C. H. Traue, S. Filipic, M. Weber, H. Hoffman, "SIMPLEX - Simulation of Personal Emotion Experience - A new Model of Artificial Emotions," University of Ulm, Medical Psychology, Institute for Media Computing, Germany, 2008

[5] J. Dias, S. Mascarenhas, A. Paiva, "FAtiMA Modular: Towards an Agent Architecture with a Generic Appraisal Framework," University of Lisbon Tagus Park, Portugal, 2011

[6] P. Gebhard, "ALMA: Layered Model of Affect," in AAMAS'05: Proc. of the 4th Inter. J Conf. on Autonomous agents and MAS, New York, USA, 2005

[7] C. Becker-Asano, "WASABI: Affect Simulation for Agents with Believable Interactivity," PhD thesis, F. of Technology, Uni. of Bielefeld, 2008

[8] A., Mehrabian, "Pleasure-Arousal-Dominance: A General Framework for Describing and Measuring Individual Differences in Temperament," In Current Psychology, 1996

Part III
Code Analysis

Chapter 5

Frequency-rank Plots in Analysis of Machine Code

1. Introduction

Since 1949, the original work published by George Kingsley Zipf [1], an American linguist, became extremely popular among scientists from many different research fields. In brief, Zipf's work describes relation between an item occurrence frequency and its rank in a set of elements. This relation roughly obeys the power law given by following formula:

$$f(r) \propto \frac{1}{r^{\alpha}}, \tag{1}$$

where *r* is terms rank, *f* states for frequency and α is a real number, close to one for many natural languages. Of course *roughly obeys* is not very precise term, and so called Zipf's Law was often criticized for not being tested as precisely and thoroughly as it should. Mandelbrot altered the formula, proposing the following equation to describe phenomenon discovered by Zipf [2], [3]:

$$f(r) \propto \frac{1}{(r+\beta)^{\alpha}}, \tag{2}$$

where β is a real number. He also proposed parameter values, namely $\alpha \approx 1$ and $\beta \approx 2.7$ for data originally used by Zipf. But even with this alteration, caution must be used when dealing with power laws in general, not only Zipf's Law, as stated in [4] and [5]. And although researchers found term frequency distributions following power-law behavior similar to Zipf's Law in many different areas, for instance in music [6], computer systems [7], [8], Internet [9] and many, many more, the aim of this paper is not search for another power-law occurrence, but to use it as guideline to show similarities and differences between certain datasets – which can be applied to i.e. forensics or intrusion prevention in computer systems.

2. Machine code as a way to communicate

Zipf proved mathematically that natural languages follow certain rules when it comes to frequency-rank distributions. It is not in the scope of this paper to discuss whether those

distributions are Zipf-Mandelbrot, Pareto, log-normal or any other class – more important is to test if there are certain patterns that emerge from different technologies or setups.

This is important question, because this would shed some light on the underlying processes that made such phenomenon possible. One can observe rules that govern frequency-rank distribution in a natural language. But human brain is too complex to just use reductionism to isolate certain functions and mechanisms and analyze their impact on those rules.

But there are another datasets similar to language – in a way that they are not sets of random values, but values that carry useful information. Those datasets, freely available for everyone, are computer programs. And it may be way too simple to use processor as model for a brain, but if data designed to be processed by the computer and by the brain shows some similarities in certain cases, it could be possible that those similarities arise from similar processes. It could be a lot easier to find those processes in computer programs, simply because we know exactly what steps were taken from algorithms design and implementation to final form of the machine code.

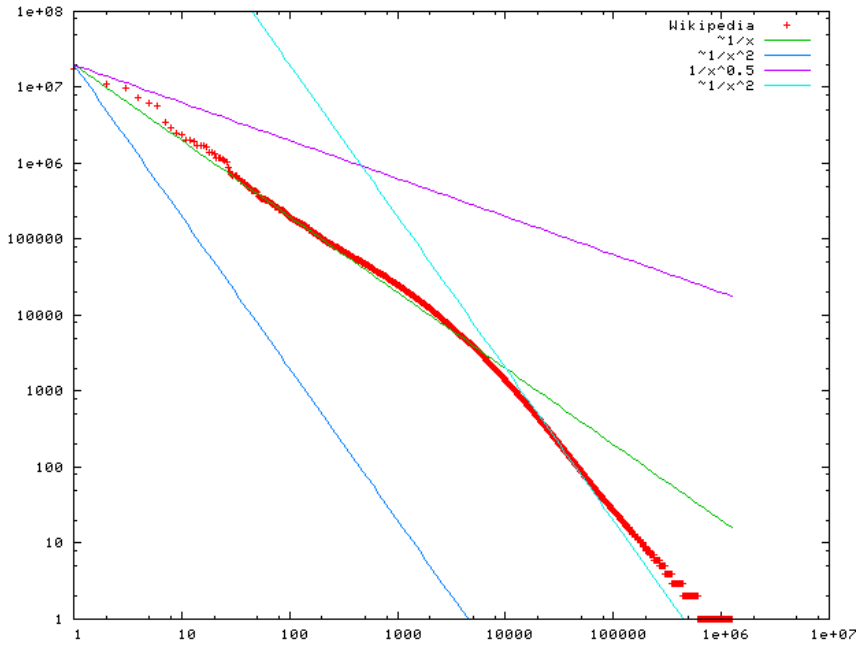

Figure 1. Frequency-rank plot for word frequency on Wikipedia, 26 November 2006. Picture published by Victor Grishchenko

Of course this is broad and open question whether all those assumptions are true – but one can only validate such statement by making appropriate steps. Testing frequency-rank plots of computer executables were one of them.

Figure 1 shows how Zipf law behaves in real, spoken languages. The reference comes from Wikimedia Foundation, which has access to quite big dataset. Note the change in

characteristic after the first 6000-8000 words. This supports the results derived from [9], that divide data into two parts – so called *limited lexicon and unlimited lexicon, which behave differently when it comes to frequency-rank statistic. It is a consequence of the very nature of language – it exhibits a kind of optimization, where most ideas can be expressed with a limited number of terms – the limited lexicon. Compilers also try to achieve different kinds of optimization, so machine code could exhibit similar behavior.*

3. Research

3.1. The data

There were three sets of data used in the experiments. First dataset consisted of 14 executable files. Four were written in Java, and the rest were C/C++ programs for both Windows and Linux.

Second dataset consisted of 68 dynamically linked libraries of different sizes and purposes, compiled for 64-bit version of Windows operating system, written mostly in pure C or C++, with few ones written in C#.

Third data set was a bit different. It consisted of executables of different nature than the first two sets – namely installers, .exe files for Windows and .rpm packages for Linux, both for 64-bit systems. They were analyzed because of the different purpose and architecture from application and libraries used to perform tasks other than installing certain software. They also utilized compression mechanisms that made them an interesting case.

3.2. Methodology

Binary read was performed on all the files analyzed. The data was then unpacked to 16-bit format (two bytes – a word) and occurrences of different words were counted.

The 16-bit format was chosen because of the resulting corpus size. As stated in [9], original Zipf's Law with proposed α parameter close to 1 was observed for most frequent words (below rank 6000 – called *limited lexicon*), and different slope was observed for the so called *unlimited lexicon*. Knowing that the limited lexicon has about 6000 words, it was sufficient to use ten times as much "words" for the whole experiment. In fact most of the files analyzed weren't even using all the possible bits combinations, even files as big as 120 MB (that is about 62 million "words" long).

Smaller unpacking formats (one byte and 4 bits) were also tested. 4 bit format was way too short to demonstrate any statistically significant results, and the results obtained for one byte unpacking are presented at the end of this section.

Longer unpacking formats (3- and 4- byte long) were problematic due to high memory usages, but are planned as a further research after optimization of the tools used.

3.3. The results

Obtained results show some interesting insights on the nature of machine code. Figure 2 and 3 illustrate different parameters and setups with one common attribute –

programming language. Both were compiled as different types of files – Dynamically Linked Library on Figure 2 and executable file on Figure 3. Both were designed for different operating systems – on Figure 2 for Windows and on Figure 3 for Linux. Both were of very different size – about 1 MB on Figure 2 and over 100 MB on Figure 3. Yet both are very similar when it comes to frequency-rank plots. Most distinctive difference is the tail area on both Figures, where smaller files exhibit many words of small frequencies across few ranks, and larger files exhibit clear cut-off close to the least ranked words.

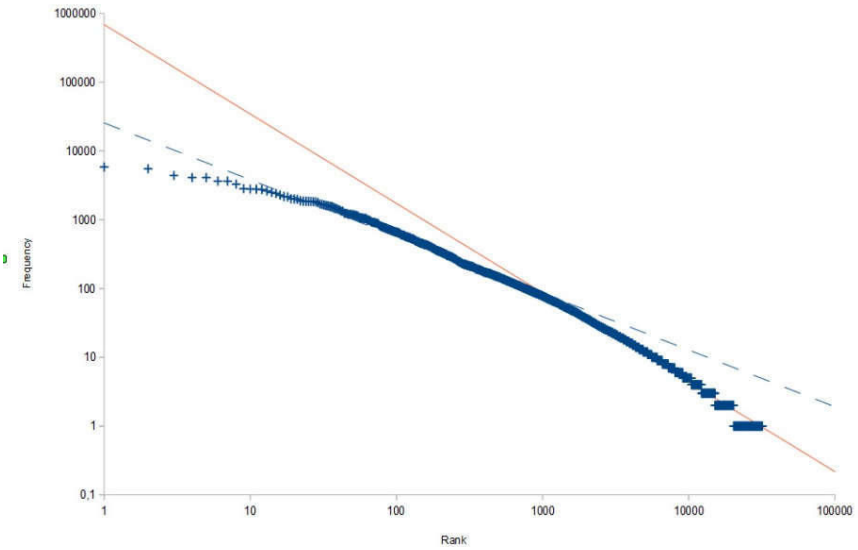

Figure 2. Frequency-rank plot for a DLL file written in C++ for Windows with file size of 1074kB. Straight line has slope of $1/x^{1.3}$ and dashed one $1/x^{0.85}$

Both figures illustrate applications written in C/C++. All files written in this languages analyzed in the study have very similar frequency-rank plots. They also resemble Zipf Law the most, with two areas of different slopes (*limited lexicon* and *unlimited lexicon*, as stated in [9]). All samples obtained from C# and Java were shaped differently, which led to conclusion that there can be a process or design characteristic of C/C++ that allows such behavior. One of the possible answers can be optimization-heavy compilation process for those languages, but this topic will be addressed in detail in further research.

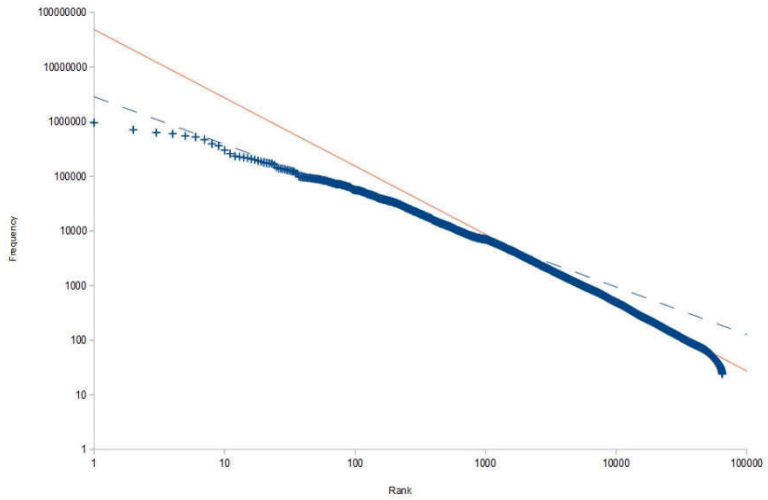

Figure 3. Frequency-rank plot for a executable file written in C++ for Linux with file size of 114 MB. Straight line has slope of $1/x^{1.25}$ and dashed one $1/x^{0.87}$

Statistics for applications written in C# had a bit more complex shape, with the middle region far from being a straight line, and thus further away from pure Zipf's Law. It is unclear why the statistic looked so different, but one of the possible explanations is that C# is a managed language with different principles when it comes for compilation – especially with different approach to optimizing the code. All results from programs written in C#, with different sizes and types of the file, were almost identical, with small differences in the middle region. Figure 4 illustrates one of the C# example plots.

Programs written in Java were the most surprising ones. All files in the study had the shapes of the statistics almost identical with each other, but quite different from the ones written in C/C++ or C#. The middle section of the plot was far from any straight line, and the tail section was relatively flat compared to the rest of the programming languages used in the experiment, with sharp cut-off at the end. It is also worth noting that the tail in this case was a lot heavier. Another interesting feature are the low differences between most frequent words, possibly coming from the Java byte code features, which could be also interesting topic for further research. Due to Java byte-code representation in compiled .jar files it is unclear what kinds of code optimization are performed and if they influence the statistics.

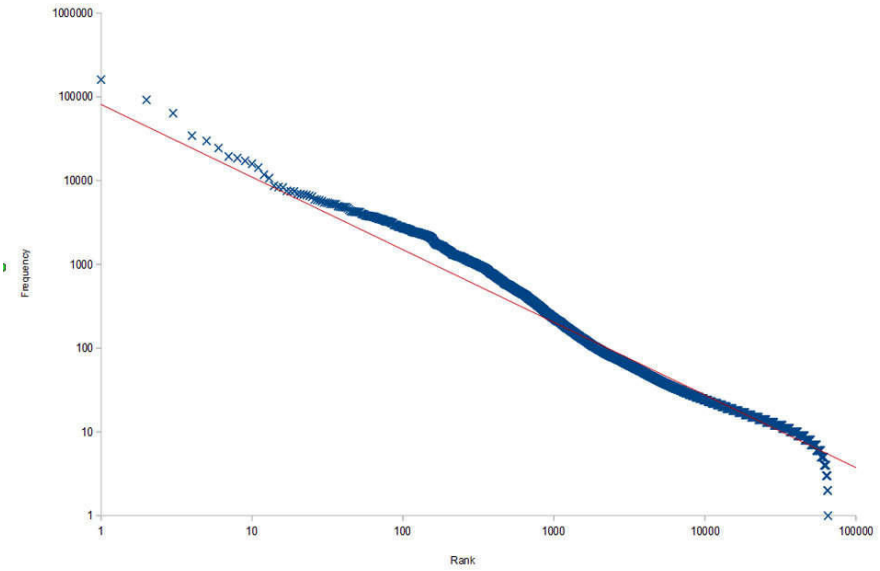

Figure 4. Frequency-rank plot for DLL written in C# for 64-bit Windows with size of 5,4 MB. The straight line has slope of $1/x^{0.86}$, but in this case should be treated only as a reference

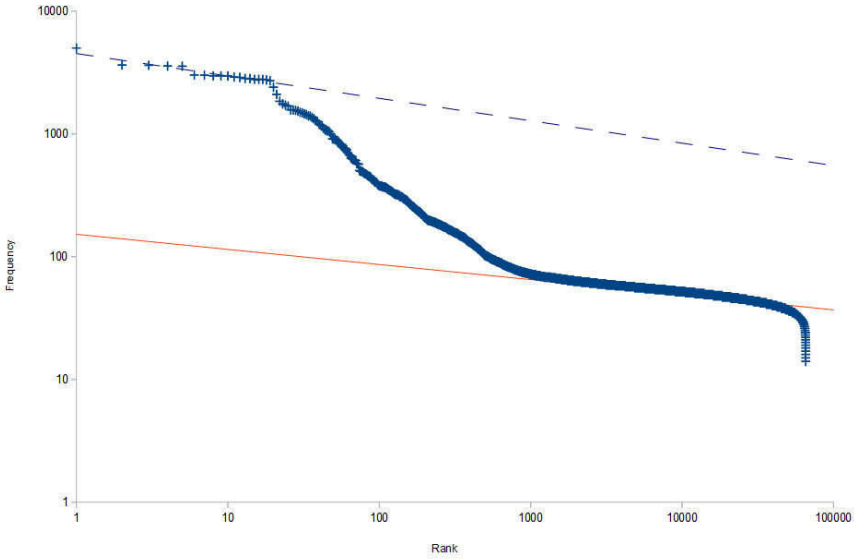

Figure 5. Frequency-rank plot for application written in Java with file size of 5,6 MB. The straight line has slope of $1/x^{0,123}$, and the dashed one of $1/x^{0.186}$

The statistic shown below should be treated only as a kind of reference. It is one of the statistics of the installers group – this one is a Linux .rpm package. The difference between this statistic and the rest of the experiment is clear – there are many words that occur with similar frequency. This time the mechanism underlying such behavior can be pointed with high probability – all the installers use compression methods that hide the details of the files inside. It is certain that compression is involved, because most compression methods base on the fact that if one can make the entropy of the file higher, one can also contain the same information in shorter file. So, when the bytes are packed, the entropy rises making frequency-rank plot as flat as this depicted on Figure 6. All installers share this characteristic.

They could also contain files quite different from executables that should be researched further with the method used in this experiment. It is still clear that, although being also an executable files, those from the last researched groups are quite different from the other. It should be also noted that Figure 6 is a statistic for the installer package of the application from Figure 3. It was extracted from this package, and yet the resulting statistic differs greatly.

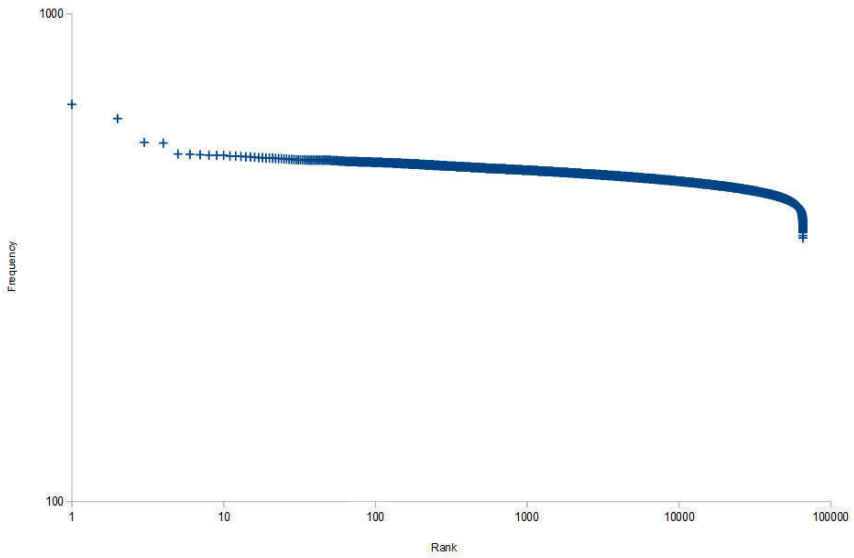

Figure 6. Frequency-rank plot for .rpm installer package for 64-bit Linux with size of 55,4 MB. Note the scale – frequencies are distributed very evenly.

3.4. Shorter unpacking

During the experiments, shorter unpacking formats were also used. As mentioned in section 3.2, one byte unpacking was also tested, with a degree of success. The lexicon was too short to compare with Zipf's Law in general, but power-law patterns were found on all datasets with this unpacking format. Figures 7, 8 and 9 illustrate the results, for the same examples as Figures 3, 4 and 5 respectively.

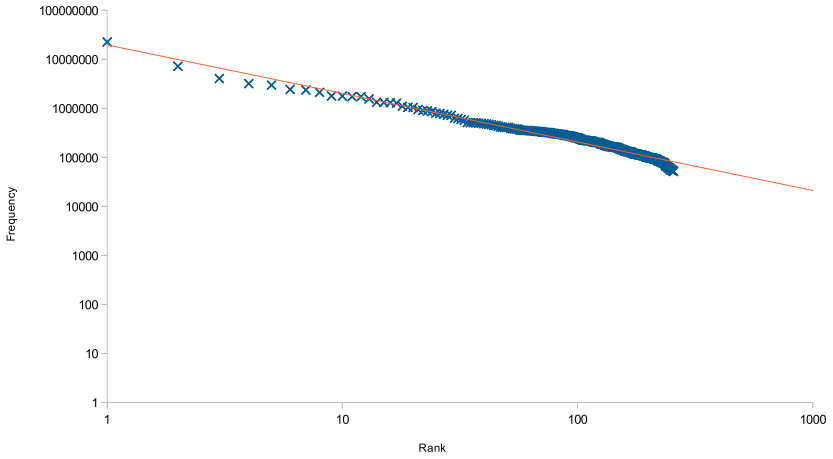

Figure 7. Frequency-rank plot for the same executable as Figure 3 (written in C++), with one byte unpacking format. The straight regression line has slope of $1/x^{0.989}$

Interestingly the C++ files, with unpacking format of one byte almost perfectly follow the inverse power law relation with exponent close to one. All C++ files analyzed followed such relation with differences in exponent at less than 0.05. It could be implied by the small lexicon size, that hides many of statistical properties of machine code in question.

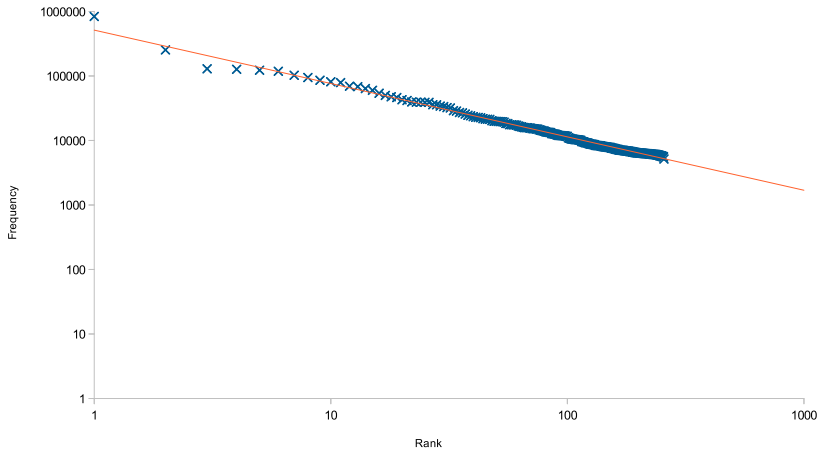

Figure 8. Frequency-rank plot for the same library as Figure 4 (written in C#), with one byte unpacking format. The straight regression line has slope of $1/x^{0.82}$

The results for C# yield similar statistic, with the slight difference in exponents. As for C++ files, all examples written in C# share similar values of the exponent, all close to 0.8. It is worth noting that the distinctive characteristics of the statistics on Figure 4 are lost in comparison to Figure 8.

It should be also noted that there were no examples in the dataset which exponent didn't match the technology used to write the file in question. Whether it is just the property of the examples used or it is a universal pattern should be focused in further research, but it can be promising mechanism in forensics – mostly because the algorithm used is linear, with complexity O(N), where N is the size of the file.

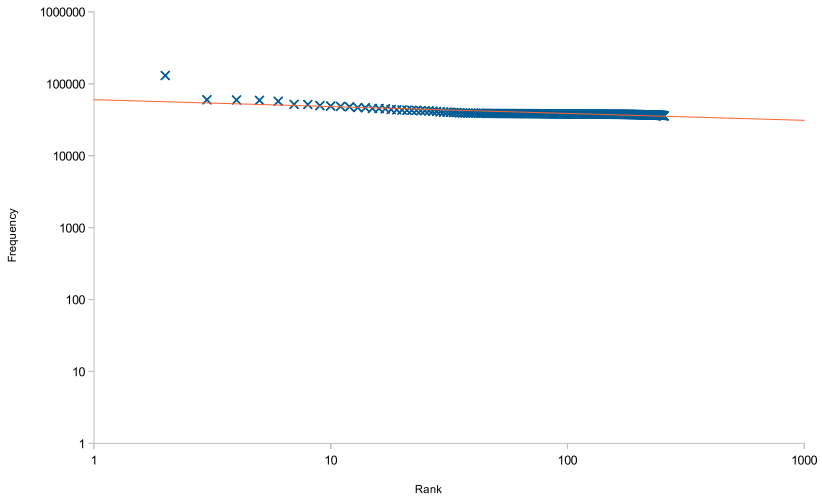

Figure 9. Frequency-rank plot for the same library as Figure 5 (written in Java), with one byte unpacking format. The straight regression line has slope of $1/x^{0.09}$

Analysis of the Java byte code also revealed interesting pattern. Figure 9 shows the results for Java application, the same as on Figure 5, but with one byte unpacking format. This statistics also follows inverse power law relation closely, but it has very low differences between subsequent data points. It is no doubt characteristic of the Java byte code, because all analyzed files written in Java yield similar characteristic, with exponents below 0.12. Java is compiled only indirectly, and it is probably affecting the shape of the plot.

As with the longer lexicons, again C++ is closest to Zipf's Law for natural language. The same conclusions emerge – it is probably due to heavy optimization in compile time. There were some experiments designed to test this observation and are described in the next section.

4. Conclusions and further research

The experiment presented shows that there are patterns that emerge from statistics of frequency distributions in machine code. The most obvious reasons of differences comes from programming language used. Some of the patterns are far from simple power law relation, but are independent from file size, operating system and whether it is a executable or library.

Further experiments should focus on differences between compilers and their output. It could be interesting to understand why C/C++ resembles natural language most when it comes to Zipf's Law and does it have something to do with high level of the language optimization. Those problems bear close resemblance to emergent properties of complex systems, where simple rules lead to very complex behavior. Maybe it is possible to understand if similar simple rules lead to such complexity as natural language. Even if not, presented research could be a part of "statistical fingerprint" of an application that could reveal some of facts about unknown file – which in turn can help in computer forensics and treat prevention.

As for the experiments with optimization, frequency-rank statistic test on the same executable, but compiled with different optimization settings could shed some light on the subject. Question about how much optimization is affected by this setting is an open one, and could be also an interesting research topic.

References

[1] G.K. Zipf, Human behavior and the principle of least e ort, Addison-Wesley Press, London, 1949
[2] B. Mandelbrot, An informational theory of the statistical structure of language, Communication theory (1953), 486-502.
[3] B. Mandelbrot, On the theory of word frequencies and on related markovian models of discourse, Structure of language and its mathematical aspects (1962), 190-219.
[4] M.P. Stumpf, M.A. Porter, Critical truths about power laws, Science 355 (2012), 665-666.
[5] S.T. Piantadosi, Zipf's word frequency law in natural language: A critical review and future directions, Psychonomic bulletin & review 21 (2014), 1113-1120.
[6] B. Manaris, J. Romero, P. Machado, D. Krehbiel, T. Hirzel, W. Pharr, R.B. Davis, Zipf's law, music classification, and aesthetics. Computer Music Journal, 29 (2012), 55–69.
[7] Y.S. Chen, Zipf's law in natural languages, programming languages, and command languages: the Simon-Yule approach, International journal of systems science 22 (1991), 2299–2312.
[8] G. Concas, M. Marchesi, S. Pinna, N. Serra, Power-laws in a large object-oriented software system. Software Engineering, IEEE Transactions on, 33 (2007), 687–708
[9] R. Ferrer i Cancho, R.V. Solé, Two Regimes in the Frequency of Words and the Origins of Complex Lexicons: Zipf's Law Revisited, Journal of Quantitative Linguistics 8 (2001), 165–173

Chapter 6

Iterative Data Flow Analysis for Code Change Propagation in Java Programs

1. Introduction

Understanding the code change consequence, its propagation through a program code is crucial for the software industry. The change scope estimation impacts decisions at different levels of software development: programmer (less invasive vs more generic), tester (identify possibly changed responses), management (fund/time allocation), sponsor (go/nogo decision), etc. Mission critical software requires high level of certainty that the change does not introduce the risk of inconsistency of logic and data flow, degradation of safety, as well as performance, security breaches, etc. Current techniques of the scope change estimation rely on human expertise and (an often huge) number of test experiments. A tool which, admittedly, excludes some parts of a program from being impacted by a change, would bring many advantages: for example, it would eliminate requirement of execution of those tests which are verifying not-impacted parts of a program.

The paper builds on the previous Author's work [4], whose general idea has been a tool which takes as input two versions of a program code, and outputs places where the program code may behave differently between versions. The result of the tool should be comparable to an analysis performed by a programmer not knowing a specific domain. The paper demonstrates the capability of creating a tool.

In particular, the paper focuses on an early analyzer implementation, resolving a fixed point.

Section 2 introduces the concept of an active *context*, which operates on bytecode, a low-level programming language, but, which benefits from the source code being a structural program. At the beginning of structures (e.g. fork operations) alternative contexts are being created, and they are joined at the end. No preliminary analysis is performed, an operation may be part of a loop, and the analyzer is not aware of that, as backward jump operation occurs later in the code. The bytecode operation is analyzed one by one, and a control flow graph is built on the fly.

Section 3 explains the internal analyzer's data structure, fragments of the analyzer's code in Prolog, and its behavior in simple examples. Section 4 discusses the result of iterative analysis of the Greatest Common Divisor function. Section 5 characterizes presented algorithm in the context of data flow analyzers. The paper is shortly summarized in Section 6.

2. Active Context Concept

Context is a set of data flow terms, a result of basic block (linear sequence of program instructions, [1]) processing. *Context* is in the directed acyclic relationship with other, parent, *contexts*. The structure of the *contexts* graph reflects the shape of a structural program. When a new structure begins, the analyzer creates new *contexts* and processes every branch of control flow in a separate *context* with the same parent/s: the same knowledge about data coming to the structure. When the structure analysis is finished, all branches are finished, the new *context* is created with all branch *contexts* as parents. This mechanism is not recognizing structures explicitly, it just follows bytecode, low-level language statements. A collection of them may reflect structural program construct: `if-else`, `for`, `while`, method call, etc. but the analyzer does not identify them. The analyzer relies on the fact that operational stack is statically known [2], which makes the concept limited to structural programs.

The analyzer requires managing of data flow, and control flow, information. Data flows are stored as terms, control flows as *contexts*. When a data-driven operation is being analyzed, a new term of data flow is created, and added to the active *context* (it is never removed). When analyzing a flow control operation, new contexts are created, with current *context* as a parent, and stored in the awaiting *context* holder with a target address assigned. When analyzer reaches the target address then awaited *contexts* are activated. When more *contexts* are assigned to a certain address then a new active *context* is created and those, waiting on the address are added as parents.

Listing 1. If-else and Loop bytecode constructs schema

```
a0: ...
a1: if_xx a4
a2: ...
a3: goto a5
a4: ...
a5: ...
```

```
a0: ...
a1: if_xx a4
a2: ...
a3: goto a0
a4: ...
a5: ...
```

Let's consider Java bytecode if-else construction from the left side of Listing 2: `a1: if_xx a4` opcode is met and data flow term (stacked values to operation `if_xx`) is created and added to current *context*. Two *contexts* are created with current *context* as a parent. The first for matching condition criteria and awaiting address is set to `a4`, and stored in an awaiting contexts holder. The second, for not-matching condition criteria, becomes active. When operation `a3: goto a5` is reached, then an active *context* is moved to the awaiting contexts holder with address `a5` assigned. Processing of the operation at address `a4` activates previously created *context* (for matching condition criteria). When address `a5` is reached, two contexts stay active: a new context is created with those two *contexts* as parents.

Loop construction from the right side of Listing 2 differs from the if-else construction only by the target address of the `a3: goto` instruction. This jump back causes that operation `a1: if_xx a4` becomes an exit loop condition, but the analyzer does not need to be warned of it. It follows jump target address till a fixed point is stable (no new data

flows are identified). Every iteration creates *contexts* by the same rules, re-visited operations are in-lined and distinguished (see Section 3.4). When the fixed point is stable, analyzer skips the backward jump and reaches address `a4`, on which multiple *contexts* are waiting (every iteration was creating new ones). Hence, new *context* is created with the following parents: 1st iteration exit loop condition met, 2nd iteration condition met, and so on till the th iteration condition met *context* (*n* is the number of iterations the analyzer has performed to achieve a stable fixed point). As an analyzed code is limited, the number of iterations is finite. Even analyzed data is cyclical, or the loop is infinite in runtime, the analyzer finishes the analysis in a finite number of iterations and with a directed acyclic *contexts* graph. Section 4 presents the *contexts*' acyclic relationships through the example of the Greatest Common Division function, in Table 1, and its cyclical data flow, in Table 2.

Apart from simplicity, the solution brings with it a disadvantage: the analyzer does not differentiate a *n*th time visited loop block of instructions from a block repeated *n*-times in source code. The resu of analysis (data flow term sets) will be the same. From the point of view of static analysis it is not mistake, but from the point of view of a change propagation analysis, it is not correct. This should be taken into account in future research.

3. Analyzer implementation

Listing 2 presents the main loop of the analyzer. Every bytecode operation goes through procedures: an active *context* selection, processing data flow, processing fork and processing jump statement. If an operation is not applicable for a certain type of processing, then it responds with no change (e.g. operation `iadd` impacts operational stack only - procedure `process0p` , operation `ifne` impacts operational stack and control flow - procedures `process0p` and `processFork` , etc.).

Listing 2. Process Method Body

```
%processBlock(+ProcBody,+MethodBody,+OpStack,+JumpStack,
%             +MethodStack,+ContextsIn,-ContextsOut)
processBody([],_,_,_,_,A,A).
processBody([Bc|BcRest],Body,S,J,M,Ain,Aout):-
selectContext(Ain,Bc,[Ctx1|Awaiting]),
processOp(Bc,S,S1,J,M,Ctx1,Ctx2),
processFork(Bc,J,[Ctx2|Awaiting],A3),
processJump(Body,Bc,BcRest,Body1,J,J1,A3,A4),
processBody(Body1,Body,S1,J1,M,A4,Aout).
```

Procedure `processBody` operates on the following data structure:

- ProcBody, MethodBody: [BC] - array of bytecodes,
- BC: bc(PC,opcode) - bytecode is a pair of program counter and opcode (e.g. `iadd`, `bipush`),
- OpStack: arrays of operations pushed onto the operational stack,

- ContextsIn,ContextsOut: [ActiveContext|AwaitingContexts] - the first element is an active *context*, the rest are pairs of address and *context* - awaiting *contexts*

Procedure `selectContext` takes contexts holder `Ain` and processed operation `Bc` , and based on the operation's address responses with an active context `Ctx1` . Procedure `process0p` changes operational stack from `S` to `S1` , and adds terms of data flow to active *context* `Ctx2`. Procedure `processFork` creates new *contexts* and stores them in context holder `A3` . Procedure `processJump` responds with a new array of operations, which are left to be processed in `Body1` .

3.1. Processing Fork Statement

Processing the fork operation is not the first step in the algorithm, but gives an overall perspective on the concept of the active *context*. Opcodes are processed within a certain environment, called *context*. Lookups and data flow terms creation are performed on this single, active *context* (and its parents). Control flow fork operation, typically one of `if` opcode (`ifne`, `ifge`, `if_icmpeg`, etc.), but also exception prone operations like `putfield`, `invokevirtual`, `idiv`, etc., create (usually) two separate *contexts* , making an active *context* as the parent for both. Newly created *context*s are named with a condition result, addressed with a jump address and stored in an AwaitingContexts. The target address is one of the following: jump address of `if` opcode, the catch section address within current method, special address '-2' meaning that the method exists with an exception, special address '-1' meaning the method exits: operation `return`.

Operational stack `0pStack` is not a part of the *context*, as data stored on it do not follow fork targets: opcodes `if` restore the stack to `zero size` (for structural programs) and Jvm runtime exception handling mechanism clears the stack, before transferring control flow to a catch/finally section.

Listing 3. Processing Fork Statement

```
processFork([Bc,bc(PC1,_)|_],C,[Ctx|Awaiting],Aout):-
  Bc=bc(PC,Con),isCondition(Con,T),
  Aw1=awaiting(PC1,ctx(d(Bc,C):false,[],[Ctx])),
  Aw2=awaiting(T  ,ctx(d(Bc,C):true ,[],[Ctx])),
  Aout=[null,Aw1,Aw2|Awaiting],!.
processFork(_,_,A,A).
isCondition(C,T):-
  memberchk(C,[ifge(T),ifgt(T),ifle(T),iflt(T),ifne(T),
    if_cmpge(T),if_cmplt(T),if_cmple(T)]).
```

Listing 3 presents fork processing procedure for conditional statements: two *contexts* are created `Aw1`, `Aw2`, with addresses of the next opcode `PC1`, and the target of conditional jump `T`. The current *context* is nulled.

3.2. Context Selection

Context selection is the first operation of every processed bytecode. When more then one *contexts* are awaiting on the processed bytecode address then the new *context* is created with multiple parents: *contexts* awaiting on this address. In this way the single *context* is always activated. A *context* may have single, or multiple parent *contexts*.

3.3. Processing Opcode Data Flow

3.3.1. Stack to stack data flow

Predicate `process0p/7` is responsible for a processing of a single data operation. It modifies OpStack and ActiveContext. Opcodes are divided into groups of stack only operations: taking two stack arguments to push the result into stack (`iadd`, `imul`, etc.), taking single element to push the result into the stack (`ineg`, `i2b`, etc.), and zero arguments, pushing constant value on stack (`iconst_1`, `bipush`, `ldc`, `new`, etc.)

Listing 4. Stack to Stack Flow

```
processOp(Bc,[V2,V1|S],[[d(Bc,C)]|S],C,_,ctx(N,DF,P),ctx(N
    ,[df(N,d(Bc,C),V1,V2)|DF],P)):-
  twoArgRetStackBc(Bc),!.

twoArgRetStackBc(Bc):-
  Bc=bc(_,C),memberchk(C,[iadd,imul,dadd,dmul]).
```

Listing 4 illustrates processing of a two argument stack operation: `V1`, `V2` are taken from the operational stack, single element array `[d(Bc,C)]` is pushed back and data flow element `df(N,d(Bc,C),V1,V2)` is added to the active *context's* data flow table.

3.3.2. Stack to condition, stack to variable data flow

Opcodes that read operational stack elements and do not push a result back on the stack are: control flow operations (`if_icmpge`, `ifnull`, etc.) or memory manipulation operations (`putfield f`, `store 1`, `aastore`, etc.). The analyzer manages the control flow information by the active *context* concept. Retrieving memory state is achieved by a lookup on this hierarchical *context* structure.

Listing 5. Stack to Operation Flow

```
processOp(Bc,[V2,V1|S],S,C,_,ctx(N,DF,P),ctx(N,[df(N,d(Bc,
    C),V1,V2)|DF],P)):-
  twoArgStackBc(Bc),!.

twoArgStackBc(Bc):-
  Bc=bc(_,C),memberchk(C,[ifge(_),putf(_,_)]).
```

Listing 5 depicts the same behavior of data flow creation as Listing 4 with the difference that the result is not pushed back on the operational stack.

3.3.3. Memory State

Memory state read operations (`iload 2`, `getfield f`, `aaload`, etc.) are linked with the corresponding memory write operations (`istore 2`, `putfield f`, `aastore`, etc.). On memory read operation the analyzer performs lookup of operations in the active *context* (and its parents) to locate those, whose results flow to the corresponding write operation. It means that when `iload 3` is met, then operations flowing to `istore 3` are looked for and pushed onto the analyzer stack.

The lookup method maps function *P*, specified by Eq. (1) of [4], to the *context* lookup prev procedure in Listing 6. The element is located when it exists in a current *context's* data flow table: `inDataFlow(DataFlowArray,E)`, otherwise it is searched for in parent contexts: `prev(ParentContexts,E)`. The neighbor *contexts* are searched for using the alternate solution mechanism, the or: `(;/2)` predicate: `prev(Contexts,E)`. The result of the lookup procedure is the table of all elements matching predicate `prev/2` collected by build-in predicate `setof/3`: `setof(F,prev([ActiveCtx],F,Result)`.

Listing 6. Lookup Procedure

```
prev([ctx(_,DataFlowArray,ParentContexts)|Contexts],E):-
  (inDataFlow(DataFlowArray,E) ->
    true;
    prev(ParentContexts,E)
  );
  prev(Contexts,E).
```

$$P(f_\alpha, F, \mathbb{F}_c) = \{f_x, f_x \in F \wedge \{f_x \to f_1, ..., f_n \to f_\alpha\} \subset \mathbb{F}_c \wedge \{f_1, ..., f_n\} \cap F = \emptyset\} (1)$$

The analyzer extends local variable opcode `store 1` with a method indicator: `store(Method,L)` to distinguish variables of different methods. It is important because the method call (one of `invoke` bytecode) is inlined. It is operated within the same *context*, opposite to JVM, which operates local variables on separate JVM's frames. Another solution, presumably more efficient, would be maintaining separate holders for local variable *contexts*, but an unified approach for local variables and object fields selected.

To find all possible values flowing to a certain memory the analyzer requires explicit data initialization. JVMcompiler guarantees that every local variable is initialized [9], `prev/2` finds proper results. Opposite to local variable initialization, fields initialization is not always explicit, `prev/2` may not find an initial value flowing to a field. In consequence, the analyzer would not gather that a certain field's value is assigned to e.g. `null`, and responds with the missing `null` flowing to other operations. To resolve this discrepancy, the analyzer assigns initial values to non-static fields of the class being instantiated, and its super classes accordingly (not implemented contemporaneously, at the time this paper is prepared).

3.3.4. Object references

The analyzer processes bytecodes one by one within a single active *context*, distinguishing re-visited operations by a loop index (see Section 3.4), which make every processed operation unique. This uniqueness allows you to identify objects by their defining new operation. This assumption is very convenient, and facilitates many aspects of analysis like: sharing, reachability, cyclicity [7,6,3,5], etc. Two references point to the same object only if they are the same new operation: `d(bc(PC,new(ClassName),LoopIndex)`.

Taking into account the above, object field assignment is realized by the data flow of two stack values (object's reference and field's value) to the `putfield` operation. It is processed in the same way like any other two argument operations, shown in Listing 5. Locating possible values of an object's field, is achieved by lookup of data flown to `putfield` with matching reference (new operation) of the analyzed operation `getfield`.

3.3.5. Method invocation

Method invocation bytecode (`invokespecial`, `invokevirtual`, `invokestatic`, etc.) processing is consistent with the active *context* concept, it does not introduce new rules. Based on the method's signature, a number of parameters are taken off the stack and converted to local variables - `store(Method+1,L)` (+1 is explained in Section 3.3.3). When the invoke bytecode is not `invokestatic`, then an additional value, object reference, is taken from the stack. Target methods are resolved based on classes specified in the object's reference defining operations `new(ClassName)`, class hierarchy, call type (static, virtual), JVMresolving method rules [9]. For every object reference value (operation `new(_)`) single method is resolved and processed (`processBody/7`) within its separate *context* (see Section 3.1). Or an exception context is created on the object's reference value `null`.

Finally, the result (address '-1':`return`) *contexts* of processing method bodies are joined (if many) with a single (actual) *context*. When the method's signature return type is different than void, then data flowing to bytecode `return` is collected (see Section 3.3.3) and pushed onto the current operational `stack`.

Exceptions method leaving *contexts* (control flow paths), grouped by special address '-2', are matched against current try-catch block definitions to re-address them with the current catch sections, or are left unchanged to mark them as leaving the current method with an exception.

3.4. Processing Jump Statement

Three scenarios have been identified of processing the control flow operation: the forward jump, the first-visited backward jump, the re-visited backward jump. The forward or backward jump scenarios occur on an unconditional or conditional jump operation: `goto(T)`, `ifge(T)`, `ifgt(T)`, `ifle(T)`, etc.

The forward jump is identified as a situation when the jump target address is bigger than the currently processed bytecode address (T>PC at Listing 7). It causes that the active

context is moved to the waiting contexts holder with a target jump address, and the next bytecode is processed (see Section 3.2).

Listing 7. Processing Jump Forward Statement

```
processJump([bc(PC,goto(T))|B],_,B,J,J,[Ctx|A],[null,
    awaiting(T,Ctx)|A]):-
  T>PC,!.
```

The first-visited backward jump causes: jump bytecode and current *context* is pushed onto the jump stack, and processing follows the jump address: processing operation list (`ProcBlock`) is loaded, and process continues with the jump target address. Second time (and more) processed operations are distinguished by a jump counter (square brackets in the analysis result table 1).

The re-visited backward jump is identified by a processing jump bytecode matching the top bytecode of the jump stack (pushed there by the first visited backward jump). The current *context* data flow elements are compared to the *contexts* pushed onto the jump stack. If they differ, then the jump counter is increased and processing follows the jump address once again. This process is repeated till no new data flow elements are identified, *context* and counter are taken from the jump stack, the next bytecode is processed.

Loop indexing has been introduced to demonstrate re-visiting the same operations by the analyzer. Let's consider example operation `11: ifeq_33` indexed by inner `23: goto 10` and outer `30: goto 10 loop` at Listing 10. The analysis starts with index `[1]`, the first time the operation is visited it is marked as `11[1]` on the data flow Table 1. When the inner loop operation is met, analyzer follows target jump address (the first visited backward jump) and our example operation is now indexed with increased depth `11[2,1]`. Next visit is marked as `11[3,1]`, and the analyzer leaves the inner loop (index id 4 is skipped, jump index is `[1]`). When the outer loop operation is met, and analyzer follows jump address, the example operation is marked as `11[5,1]`. Now, inner loop creates visits `11[6,5,1]` and `11[7,5,1]`. The analyzer leaves the inner, and then the outer loop.

4. Case Study

Listing 8 presents Greatest Common Division function, and Listing 10 presents its bytecode representation.

Listing 8. Greatest Common Divisor

```
/**
 * @param(a, b) the values used to
      calculate the divisor
 * @return the greatest common divisor
      of a and b
**/
            c = c − d;
        else
            d = d − c;
    }
    return c;
}
```

```
int gcd(int a, int b) {
    int c = a;
    int d = b;
    if (c == 0)
        return d;
    while (d != 0) {
        if (c > d)
```

```
[d, d=b, return d]
[d, d=b, while(d!=0)]
[d, d=b, if(c>d)]
[d, d=b, c=c−d]
[d, d=b, d=d−c]
[d, d=d−c, while(d!=0)]
[d, d=d−c, if(c>d)]
[d, d=d−c, c=c−d]
[d, d=d−c, d=d−c]
```

Listing 9. Use-Def Chains of variable d

Listing 10. Greatest Common Divisor Bytecode

```
0: iload_0
1: istore_2
2: iload_1
3: istore_3
4: iload_2
5: ifne 10
8: iload_3
9: ireturn
10: iload_3
11: ifeq 33
14: iload_2
15: iload_3
16: if_icmple 26
19: iload_2
20: iload_3
21: isub
22: istore_2
23: goto 10
26: iload_3
27: iload_2
28: isub
29: istore_3
30: goto 10
33: iload_2
34: ireturn
```

Table 1 presents the result of Listening 10 analysis. Column **Id** keeps the analyzer's internal context identifiers, column **pId** parent context identifiers, and **Data Flow** presents data flow terms in the following format: <target bytecode>[<iteration label>](<comma separated source bytecodes of the first argument; comma separated source bytecodes of the second argument>). Context 1 contains the following terms of data flows: 1(a) (parameter a flows to `1: istore_2`), 3(b) (parameter b flows to `3: istore_3`), 5(a) (parameter a flows to `5: ifne 10`). As a consequence of conditional operation 5 two contexts are created: with id 2-for not met condition, and id 3-for opposite (waiting on address 10). Context `2` contains term 9(b) (parameter b flows to `9: ireturn`) and it is stored in the awaiting context holder for return address '-1'. Context id `3` defines data flow term to operation `11: ifeq 33` as `11[1](b)` and two (fork) contexts are created (with parent id 3): id `4` for not met condition `11`, and id `5` for met condition and

pushed for waiting on address `33`. Next processed bytecodes create data flow term `16[1](a;b)` within context id `4`, and creates two contexts: id `6` and `7`. Context id `7` waits on address `26`. Context `6` holds the first time visited `c = c - d`; Java block, defines data flow terms as variable `c` assignment `22[1](21[1])` and subtract operation `21[1](a;b)`. Then the analyzer follows backward jump operation `23: goto 10`, creates context id `8` (with parent id `6`) and increases jump index to `[2,1]`. Within context `8` data flow term is defined: `11[2,1](b)`: still the only possible value for variable `d (load 3)` is parameter `b`. The analysis finishes this, and the next iteration (labeled with jump index `[3,1]`), and leaves the intern loop, reaches address `26` on which four contexts are awaiting: `16,17,12,7` with four possible variable `c (load 2)` definitions: `a` – through context id `7` and `21[1], 21[2,1], 21[3,1]` - results of subtract operation `21`: isub of 1st, 2nd and 3rd iteration accordingly.

Table 1. Contexts and theirs data flow terms of GCD function

Id	pId	Data Flow	Id	pId	Data Flow
36	35,2		9	8	16[2,1](21[1];b)
35	34,31,26,21, 15,10,5	34[1](a,21[1],21[2,1], 21[3,1],21[5,1],21[6,5,1],21[7,5,1])	8	6	11[2,1](b)
			6	4	22[1](21[1]),21[1](a;b)
34	32,33,28,23	29[5,1](28[5,1]),28[5,1](28[1];a,21[1],21[2,1], 21[3,1],21[5,1],21[6,5,1],21[7,5,1])	4	3	16[1](a;b)
			3	1	11[1](b)
32	30	22[7,5,1](21[7,5,1]),21[7,5,1](21[6,5,1];28[1])	1		5[1](a),3[1](b),1[1](a)
30	29	16[7,5,1](21[6,5,1];28[1])	17	14	
29	27	11[7,5,1](28[1])	12	9	
27	25	22[6,5,1](21[6,5,1]),21[6,5,1](21[5,1];28[1])	7	4	
25	24	16[6,5,1](21[5,1];28[1])	33	30	
24	22	11[6,5,1](28[1])	28	25	
22	20	22[5,1](21[5,1]),21[5,1](a,21[1],21[2,1],21[3,1];28[1])	23	20	
20	19	16[5,1](a,21[1],21[2,1],21[3,1];28[1])	31	29	
19	18	11[5,1](28[1])	26	24	
18	16,17,12,7	29[1](28[1]),28[1](b;a,21[1],21[2,1],21[3,1])	21	19	
16	14	22[3,1](21[3,1]),21[3,1](21[2,1];b)	15	13	
14	13	16[3,1](21[2,1];b)	10	8	
13	11	11[3,1](b)	5	3	
11	9	22[2,1](21[2,1]),21[2,1](21[1];b)	2	1	9[1](b)

Table 2. Data Flow between bytecode operations of GCD function

target	source	target	source
bc(5,ifne(10))	a	bc(21,sub)	bc(21,sub),bc(28,sub),a,b
bc(9,ireturn)	b	bc(28,sub)	bc(21,sub),bc(28,sub),a,b
bc(11,ifeq(33))	bc(28,sub),b	bc(34,ireturn)	a,bc(21,sub)
bc(16,if_cmple(26))	bc(21,sub),bc(28,sub),a,b		

When the whole method body is processed by the analyzer, then contexts waiting on return address '-1' are picked to collect all return contexts (see context 36). During the whole analysis process, the acyclic context graph is built. However, skipping the loop index the cyclic data flow relations may appear. Table 2 shows all data flow relations between bytecode operations of Listing 10. It can be seen that the same set of data: `bc(21,sub),bc(28,sub),a,b` flow to operations: `bc(16,if_cmple(26))`, `bc(21,sub)` and `bc(28,sub)`. The result may also be compared to Use-Def chains for variable `d` at Listing 9: all possible operations flowing to operation `store 3` are: operation `bc(28,sub)` and parameter `b`.

5. Analyzer implementation

Proposed data flow analysis algorithm, comparing to other approaches, may be characterized as a naive, top-down, structure sensitive context, abstract interpreter. The analyzer may be called a naive as it processes bytecode opcodes without any preliminary analysis, a control flow and a data flow are discovered at the same time. It processes bytecode opcodes one by one, manipulating an active context only on control flow forks and joins.

The algorithm belongs to the top-down analyzers group with all theirs limitations: partial interpretation does not lead to the solution, calculations are not re-used. Opposite to an incremental approaches e.g. k-CFA [8], for which the precision increases iteratively. The limitation of the top-down approach is also the requirement of the access to a full program code but it is often not feasible, e.g. for runtime libraries (server libraries, security, etc.), native methods (input/output, environment settings, etc.). In the result, the analyzer requires mock implementations of these methods, for which bytecode is not available.

The presented algorithm converts bytecode operations into data flow relations, and once collected they are not modified during the process. The destructive update [10] is implemented as new data flows on control flow paths that overlap the previous one.

In the literature there are different approaches to the context: from the contextless, static, such as [2], 0-CFA to the path-sensitive context. The author's chosen structural limited context is based on the fixedpoint approach: it is sensitive enough to track dynamic data structures by references (not by types) but joins information coming from contradictory control flows. As a result, some paths can not be identified as unreachable and may be unnecessarily analyzed.

In addition, what makes the solution unique is the goal of the analyzer: the elimination of data flow techniques (through an operational stack, local variable, object field). Another uniqueness is the identity of the object through its creation operation `new`.

Studying how symbolic execution (constants, arithmetic, statistically-known arrays, etc.) affects the accuracy of the analyzer is part of future research.

6. Summary

The paper presents follow-up of [4], towards a tool that is able to create a graph of code change propagation. It formalizes the approach to repetitively visited blocks of code. In particular, recursion is not discussed but an iterative loop approach sounds promising for this construct too. It does not mention about the crucial construct of Java data structure, which is arrays. Arrays planned to be approximated as the object in which the index is mapped to the field name, except that the statically unrecognized element is mapped to all elements as a potential value. However, this subject requires future experiments with real examples to examine how much information about arrays is statically available.

That which differentiates the presented approach from typical static analysis is the premise of mapping the program code to the data flow representation: which differs when the modified program generates a different result for any input data. At the same time, it

allows for a generalization to conclude analysis of shape, sharing, reachability, cyclicity, etc.

The presented solution fulfills the assumption of a comparison tool: the data transfer technique does not affect the flow of data, it brings dynamic structures (and references to objects) to the same level of support as local variables, or data flowing through the operating stack. Operation new , indexed by a loop index, becomes the unique definition of an object: and it can be used for precise, iterative, analysis over acyclic control flow graph (contexts).

References

[1] Frances E. Allen, Control Flow Analysis, Proceedings of a symposium on Compiler optimization (1970), doi: 10.1145/390013.808479

[2] S. Genaim and F. Spoto, Information Flow Analysis for Java Bytecode, Proceeding VMCAI'05 Proceedings of the 6th international conference on Verification, Model Checking, and Abstract Interpretation (2005), 346–362, ISBN: 3-540-24297-X 978-3-540-24297-0

[3] Genaim and D. Zanardini, Reachability-based acyclicity analysis by Abstract Interpretation, Theoretical Computer Science 474 (2013), 60–79, doi: 10.1016/j.tcs.2012.12.018

[4] G. Kochanski, Data Flow Analysis for Code Change Propagation in Java Programs, Software Engineering: Improving Practice through Research, 18th KKIO Software Engineering Conference, 15-17 September 2016, Wrocław, Poland, PTI, Polish Information Processing Society, 2016, ISBN: 978-83-943248-2-7, pp. 187–203

[5] D. Nikoli´c and F. Spoto, Definite Expression Aliasing Analysis for Java Bytecode, Theoretical Aspects of Computing – ICTAC 2012 7521 (2012), 74–89

[6] D. Nikoli´c and F. Spoto, Reachability analysis of program variables, ACM Transactions on Programming Languages and Systems (TOPLAS) 35 Issue 4 (2013), Article No. 14, doi: 10.1145/2529990.

[7] S. Secci and F. Spoto, Pair-sharing analysis of object-oriented programs, SAS'05 Proceedings of the 12th international conference on Static Analysis (2005), 320–335 (english), ISBN: 3-540-28584-9 978-3-540-28584-7

[8] Olin Grigsby Shivers, Control-flow analysis of higher-order languages of taming lambda, Doctoral Dissertation (1991), Carnegie Mellon University Pittsburgh, PA, USA

[9] The Java Language Specication, Java SE 7 Edition, 2011, URL: http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf [accessed: 2016-02-20]

[10] F. Spoto, Mesnard F., and Payed É, A Termination Analyser for Java Bytecode Based on Path-Length, ACM Transactions on Programming Languages and Systems (TOPLAS) (2010), Volume 32 Issue 3, Article No. 8, doi: 10.1145/1709093.1709095

Part IV
Non-functional Aspects of Software Systems

Chapter 7

Selected Aspects of Security Mechanisms for Transactions in Bitcoin Virtual Commerce

1. Introduction

Electronic transactions are becoming one of the most popular ways of payment of everyday use. Even though this method of payment is accessible, easy to use and provides the user with high mobility (contactless payment with credit card), it lacks both security and anonymity. Due to the fact that electronic transactions are managed by banks and other third parties, the user cannot be sure if the data are not being changed or the account isn't disabled (bankruptcy of the bank). Furthermore, there is still possibility of reversing transactions, that is why the user cannot be sure whether he loses his money.

With the increasing demand for higher security as well as anonymity of transactions, the cryptographic cryptocurrency called Bitcoin was created. Contrary to other ways of payment, Bitcoin is not a trust-based model [6], [14]. The trust is guaranteed by a cryptographic proof, which eliminates the necessity for any third parties [3], [4], [9]. Moreover, Bitcoin network is created and monitored by the users. All existing transactions can be seen by everyone within the network, which eliminates the possibility of any harmful changes (e.g. spending same Bitcoin twice).

In case of anonymity, Bitcoin provides the user with almost full secrecy of his identity. No personal information is needed to use this cryptocurrency. Instead, the user is identified with addresses (one or more). There is no precise information about the user. The security aspects are fulfilled by the multiple cryptographic algorithms, which constitute the Bitcoin network. Most data is hashed and as in case of validating transactions, a kind of a mathematical riddle has to be solved and accepted by other users.

From the user's point of view, the security of Bitcoin transaction depends on ways of storing Bitcoins. Currently there are many possible ways to do it. They differ from one other in level of provided security let alone mobility of payment. Nowadays almost everyone uses smartphones. That is why a mobile Bitcoin wallet is getting more popular in case of everyday transactions. Nevertheless, already existing solutions can still lack sufficient security level. There is still a need for a reliable, easy to use mobile wallet, which will assure the reliable encryption mechanism or backups, what will reduce the possibility of losing savings.

The created mobile wallet application was realized using Xamarin mobile application creation software and enables such functionalities as log in, scanning QR codes (Quick

Response), performing transactions, adding Bitcoin addresses. Hashing password or confirming entered data are included in the application security aspects. The aim of this solution is to introduce sufficient security level of Bitcoin transaction from the user's point of view, which is based on the storage of Bitcoin.

The problem presented in the paper concerns the security issue of storing the Bitcoins. It focuses on a mobile wallet and its security aspects. It should be clear as well as easy to use. Furthermore, it should provide the user with the QR codes connected functions.

The presented paper is structured as follows: section 2 gives the outline of Bitcoin cryptocurrency. Section 3 deals with security aspect of Bitcoin transactions. Section 4 presents the exemplary application, mobile Bitcoin wallet that fulfils the security rules of Bitcoin transaction.

2. Idea of Bitcoin cryptocurrency

Bitcoin is an *anonymous, decentralized virtual cryptocurrency* created in 2009 by anonymous programmer calling himself Satoshi Nakamoto [3]. No central services or third parties are involved in management or flow of this currency, which means transactions are made between two Bitcoin users without going through any central authority. As all other cryptocurrency it has certain value, which is established by the system managing the flow and emission of this cryptocurrency [2], [10]. This system is based on the community of users, cryptographic algorithms and P2P (peer to peer) network. It means that Bitcoin is open for everyone and the "account" in this currency belongs only to the owner [1], [4].

In the Bitcoin network each user owns set of public and private keys, stored for example in a wallet, which are needed to proceed the transaction. If a user wants to send Bitcoins to another user, he must announce the transaction and sign it with his private key. The user, who receives the Bitcoin, accepts it with the public key. To avoid the possibility of double-spending same Bitcoins, all transactions have to be publicly announced to all other users [3]. For this purpose *Blockchain* is used. It contains information about every transaction, which was processed and is accessible for everyone. Thanks to this mechanism every broadcasted transaction can be validated by the users as a safe transaction [12]. Furthermore, each block of transactions is linked with the previous block, forming a chain.

With the increasing demand for a faster way of payment as well as the ability to perform transfer of money to other parts of the world, electronic payment system was introduced [2]. Beside obvious advantage, the international transactions are always bound with additional fees, not always small and need mostly at least one day to be transferred. What is more, they are based on a centralised system, so they depend on banks and other third parties. There is also possibility of the so called chargebacks so the user can also lose his money for the transaction [10].

In case of Bitcoin, as it was mentioned before, the value is stated by the free market. It is decentralized, so it works independently from banks or other third parties. Nothing can be changed in the system without the permission of the community – the users. No one can block your account or chargeback the transaction [2]. Furthermore, transactions

can be performed at any time to any place in the world. There are no large provision in case of transfers and no need for currency conversion.

Inarguably, the biggest advantage of Bitcoin is its anonymity, resistance to the inflation and security. All transaction can be performed anonymously – there is no information about the user (name, phone or other personal information) [3], [4]. There is a limitation for existing Bitcoins – 21 million, so after that limit is reached no new Bitcoins will issued. The whole Bitcoin system is based on cryptographic algorithms, which represents a secure way of storing the currency, so there is a small chance of losing it [2].

2.1. Idea of Bitcoin functionality

As mentioned before, Bitcoin is based on a decentralized system, so in other words it is not controlled or monitored by any bank or other third parties. In case of monitoring transactions, which are performed, *Peer to peer (P2P)* network is used [1]. By using P2P network, all its users become the "bank" – very of each user has information about all processed transactions in the network. It is so called a *Blockchain*. It contains information about all past and new transactions in the Bitcoin system [2], [4].

For the security purpose, each transaction has to be announced in the network, to prevent *double-spending*, which means that the same Bitcoin cannot be sent to two different users at the same time [6]. For example Alice wants to send a Bitcoin to Bob – make a transaction. The massage is sent to other users in this network with the information about the possibility of this transaction. Now it is the task of the network users – they have to decide, whether this transaction is correct. They check if Alice really owns that Bitcoin. If yes, the message is sent to the system that the Bitcoin can be transferred to Bob. If sufficient number of users post the same message, the transaction can be accepted by Bob. All users update their Blockchain with the information that this particular Bitcoin is right now the property of Bob and that transaction was completed successfully [2].

P2P network guarantees that the network will not be shut down or any double-spending would be done. All users monitor what happens in the network and transactions are also checked by the users. Furthermore, all transactions from the beginning of the Bitcoin system are stored in the *Blockchain*.

As mentioned before Bitcoin network is anonymous. In case of identification between users, Bitcoin addresses are used. Bitcoin address is a case sensitive identifier of 26 - 25 alphanumeric characters, which begins with 1 or 3 [6]. Each Bitcoin address consists of a pair of keys – public and private key. The owner of Bitcoins own the private key, which is used for signing transactions and cannot be revealed to other users. Furthermore, if the private key was lost, the owner will not be able to get back his Bitcoins.

2.2. Bitcoin transactions

Transaction between two users is performed, when one of them prescribes his Bitcoins from one of his addresses to the other address. For example user A sends Bitcoins to user B. Because of the fact that Bitcoin is decentralized, all information about transactions are open to other users. We can check that user A sent Bitcoins from his address to user B's address, we do not however have exact information, who is user A

and who is user B [1], [4], [5]. Figure 1 shows how the transaction looks like. Please note that the first line of text that follows a heading is not indented, whereas the first lines of all subsequent paragraphs are.

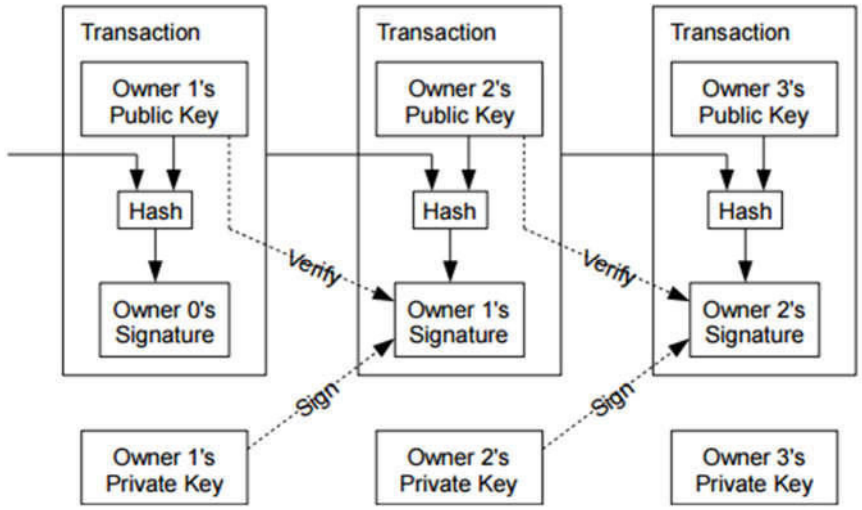

Figure 1. Chain of Bitcoin transactions [3]

Every new transaction has to point to the previous transaction. It proves that the owner of a Bitcoin got it previously from someone else or was mined (it is also considered as a transaction) [3], [5]. In other words, a user cannot send Bitcoins, which weren't received from someone else. Let imagine that Alice wants to send 0.5 Bitcoin to Bob. Firstly, she must receive 0.5 or more Bitcoin either from another user of Bitcoin network or through the mining. She receives 1 Bitcoin from her friend. Now, she can make a transaction to Bob. She has to specify the address of Bob and how much she wants to send. Because she owns 1 Bitcoin she cannot send 1.2 Bitcoin to Bob – it is guaranteed by the fact that Bitcoin network require the continuity of the Bitcoin transactions.

Alice decides to transfer 0.5 Bitcoin to Bob. She signs the transaction with her private key, ensuring the network that she is the current owner of Bitcoins. The input value will be 1 Bitcoin (so the amount of Bitcoins she received from her friend – last transaction) and the output value will be 0.5 Bitcoin (so how many Bitcoins she wants to send). The difference between these two values will be reduced by the small transaction fee and returned to the owner – Alice [1]. Figure 1 shows the input and output value.

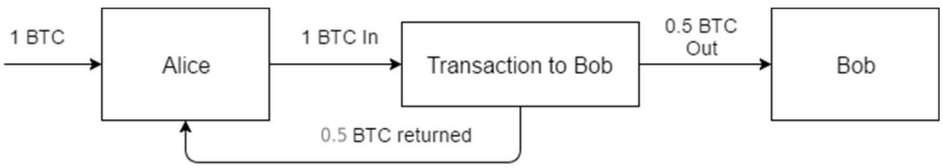

Figure 2. Transaction input and output value

2.3. Rules of Bitcoin transactions

A *proof of work* is one of the cryptographic aspects in Bitcoin network. It prevents from double spending the same Bitcoin. The block of transactions is being hashed, so no one can change anything in that block [2]. The proof of work involves generating values, as long as the value, when hashed for instance with the SHA-256 algorithm, begins with a number of zero bits defined by the difficulty [3], [11].

New transactions performed in the Bitcoin network are being stored in a block. A block is storing information about transactions and it is secured in a way it cannot be changed ever since. The number of transactions in one block depends on the number of active users in the network at a given time [1], [11]. As mentioned before, the block can be added to the Blockchain after providing the proof of work by a *miner*. Every new block of transactions should be added to the Blockchain every 10 minutes as mentioned.

Blockchain is an enormously essential aspect of Bitcoin network. It enables to determine, who is the owner of the particular Bitcoin. It is guaranteed by the ordered history of Bitcoin transactions, which is accessible for every user of this network. Every new block of transactions consists of a hash of previous block – creating so called chain of blocks. The only block that does not have the reference to the previous block is a genesis block – the first existing block in the Blockchain [5], [14]. Blocks are arranged one by one in the line and the last block has the reference to the previous block.

This chain of blocks ensures one of the security aspects of Bitcoin network. It protects from double-spending the same Bitcoin as well as prevents the modification of previous transactions [2], [14].

From this point, it is not clear, which transaction was first [2], [7]. Bitcoin network has an easy solution for such situation. This solution states that miners should work on extending the fork which is longer. Let's imagine the part of miners work on extending the block A and others the block B. Block B miners are closer to show their proof of work. The message with such information is sent to the network – the fork of block B will be longer.

Miners from block A automatically start working on the longer fork, and so do the ones from the B one [2], [13]. The block A becomes ignored, nevertheless all transactions waiting for validation in A, will be now queued in fork B [2]. Transaction in the block is accepted by the network, when it will be part of block from the longest fork and when at least 5 blocks follow it (so they are newer than the block in which there is pending transaction). Thanks to this the network has time to agree on the order of blocks.

Mining is the process of creating new blocks in Blockchain. For each validated block, miner is rewarded with certain amount of Bitcoins, which decreases every 4 years by half, and small transaction fee from other users. Miners provide with their CPU and compete with their proof of works with one other on validation of the transaction [2], [7]. Each miner has a percentage chance of winning competition equal to the percentage of CPU given to the network. For example one of the miners provides 1% of CPU of the whole network. It gives him 1% of winning the competition and receiving the reward for their proof of work [2], [14]. Such restrictions and needs for huge amount of CPU in network discourages from dishonest miners, who will have a small chance to win the competition.

Bitcoin is becoming more and more popular, nevertheless there is still lack of legal regulations. In some countries Bitcoin is considered private way of payment but in others

it is illegal. In terms of law, Bitcoin is not a real currency, since it does not fulfil the basic definition of the currency. It is decentralized, so no institution, bank or other third party controls it. There is lack of regulations, which would even consider Bitcoin as a virtual currency [1], [13].

3. Security aspects of Bitcoin

Bitcoin security is mostly based on cryptographic algorithms, which handles Bitcoin network. Inarguably, open P2P network, which controls whether there is any harmful changes in the network is also valuable aspect of its security. Furthermore, decentralized structure of Bitcoin ensures that it is not controlled by banks or other third parties, which could have impact on your account or apply any fees.

From the user point of view, security of Bitcoin is based on ways of storing Bitcoin. Today there are many ways of storing Bitcoin (*Bitcoin wallets*), but they will never be 100% sure, unless the user will be conscious of potential hazards like malicious software. It is essential to encrypt the wallet, make regular backups and keep the software up to date, to not to lose Bitcoins.

One of the most vital things in security of Bitcoin is a strong password to users' wallet. It should be long and consists of various signs, letters as well as numbers. Most of wallets do not support restoring password, so it is essential not to lose it. As mentioned before, Bitcoin uses public *key cryptography*. The user has two keys: private, which is only for the owner and public key. Message is encrypted with the public key and is decrypted with the private key of the receiver. It is used also in digital signatures, which confirms that the owner of Bitcoins is the sender.

3.1. Anonymity of Bitcoin

As it was already stated, *Bitcoin anonymity is assured by generated Bitcoin address*. It does not contain any private information of the owner of that address. Nevertheless, every address can be checked in the Blockchain and the history of transactions is available for every user. That is why Bitcoin network suggests to use new Bitcoin address every new transaction [8], [12].

SHA (*Secure Hash Algorithm*) is a cryptographic hash function, which is used in Bitcoin protocol. Right now, SHA – 256 is used in Bitcoin network for hashing blocks [7]. Hash function works basically in such a way that it takes some input data and transforms it into unpredictable as well as effectively-impossible to reverse string of 32 bytes (in SHA – 256) [8]. Each hash matches only particular input.

In case of Bitcoin such mechanism guarantees that block of transactions was not changed by anyone. Furthermore, it is used in mining, precisely saying in generating the proof of work. As it was mentioned before difficulty of proof of work (so how many numbers "0" should be placed in the beginning of the value) is adjusted every 2016 blocks, so that the new block should be added every 10 minutes [2], [12]. The miners are looking for such value that when hashed, will give the hash value with required number of 0 at the beginning. Getting back to the example, the miner will have to find hash of "Bitcoin"

so that it will begin with 0. The so called nonce is added to the value "Bitcoin". Nonce starts with the 0 and is being incremented until the value with 0 at the beginning is found [3].

ECDSA (*Elliptic Curve Digital Signature Algorithm*) is used for ensuring that funds can be only spent by the owner [12]. It also specifies the range of valid private keys, from which public key can be calculated. Furthermore, it is used in generating signature for the transaction to verify who is a current owner [7].

RIPEMD (*RACE Integrity Primitives Evaluation Message Digest*) is another hash function used in Bitcoin protocol. It converts public ECDSA key into Bitcoin address [4]. It creates shorter hash than SHA [8]. RIPEMD – 160 is right now used in Bitcoin protocol, resulting in 160-bit hash result.

Base58Check is used in Bitcoin protocol for encoding Bitcoin addresses [8]. Precisely, it is encoding binary data into printable characters of possible 58 alphanumeric symbols. With that hash function, encrypted address can be distributed between other users. That hash function states some conditions how the Bitcoin address should look like. Firstly, all Bitcoin addresses starts with the digit "1" as a hash of a public key, due to the fact that it informs that they are in a Bitcoin network. Furthermore, such address cannot contain signs such as "0", "O", "I" and "l" [4], [8].

3.2. Secure ways of storing Bitcoin

From the user perspective security depends on ways of storing Bitcoins. There are several ways of storing user's Bitcoin. Such storage is called *Bitcoin wallet*. They differentiate between one each other in terms of the mobility, availability as well as security level.

The very basic Bitcoin wallet is *Paper Wallet*. As the name says, it is a wallet in the form of paper. It provides user with high mobility, since it can be carried as normal cash, nevertheless it is highly invulnerable to any damages. Furthermore, there is a high risk of loss it (theft) without sufficient security barrier [4].

Another type of wallet is a *desktop wallet*, which is available only on user's PC. As a result it does not provide high mobility, however it is rather safe if there is no risk of malware. There is also the possibility of a *web based wallet*, which is rather mobile, nevertheless it is rather insecure, due to the fact that it depends on creators of this online wallet.

Next possibility of Bitcoin storage is *hardware wallet*. Mostly in the form of a USB (*Universal Serial Bus*) device [10]. Beside sufficient security, it is immobile, since the user cannot pay with it everywhere – it needs a USB hub. Last but not least is a *mobile wallet*. It provides the user with high mobility, due to the fact that it always available for payments. It can be also vulnerable to theft, however it is mostly secured with a strong password.

4. Mobile Bitcoin wallet

As stated before, from user's point of view the security of transactions depends on the way the Bitcoins are stored. The most mobile as well as accessible way is a mobile wallet. There are already many existing mobile wallets, which provide the user with different levels of security and they use a variety of technologies. Nevertheless, there is still lack of sufficient protection of Bitcoins. This section presents the creation process of such wallet called "*Quackllet*".

The application was created in C# using *Xamarin* mobile application creation software for Android and *NBitcoin* library for the .NET platform. It enables using *TestNet*, which is a Bitcoin network for development purposes, with which the tests of an application can be performed.

4.1. Requirements

The main aspect of the mobile wallet application is to provide the user with sufficient security. In order to achieve this goal the password to the wallet should be encrypted. Furthermore, the user should be able to confirm, whether all information is correct (in case of making a transaction). The possible usage of QR code scanning technology should be supported to avoid the possible mistakes when entering data. Another security aspect to fulfil should be the creation of a new address for every new transaction to ensure the anonymity of the user. This solution is strongly supported by the Bitcoin network.

The main functional requirements for the mobile wallet application are as follows (Fig. 3):

- verification of the existing wallet,
- access protection using a password,
- validation of the entered password,
- displaying general information about the wallet,
- navigation through the menu bar,
- scanning QR code,
- input validation,
- address management,
- performing offline backup.

As for the functional requirements, the application should verify, whether the wallet already exist. In case it is the first start of the wallet, the application should ask the user to set a password, which will be used to prevent unauthorized access. Furthermore, the check of the entered password may be needed to avoid the situations like a missclick, when the user should be moved to a login screen, in which he will be asked for the password. Having moved to the main screen, the user should see general information about the wallet (like his address and QR code).

Another requirement is the menu bar which will enable the user to easily navigate through the application. If the user wants to make the transaction, he should be able to enter the address of another user, scan the QR code or take the address from contact list. In the places, where the data is entered (the address, the amount of Bitcoins to send) there should be a validation to check whether these text fields are not blank. Afterwards, the

user will be asked if all the information is correct and the transaction will be made if the answer is yes. If the user has multiple addresses, he should be able to manage these addresses. The options like preview of the existing addresses, generating new ones or choosing the address should be supported. In the contacts option the user will be able to add new contact by either scanning the QR code or manual input. To satisfy the need for the backup function, the private keys will be saved to a file. Thanks to this the user will be able to perform an offline backup (a file containing saved private keys may be stored in the memory card).

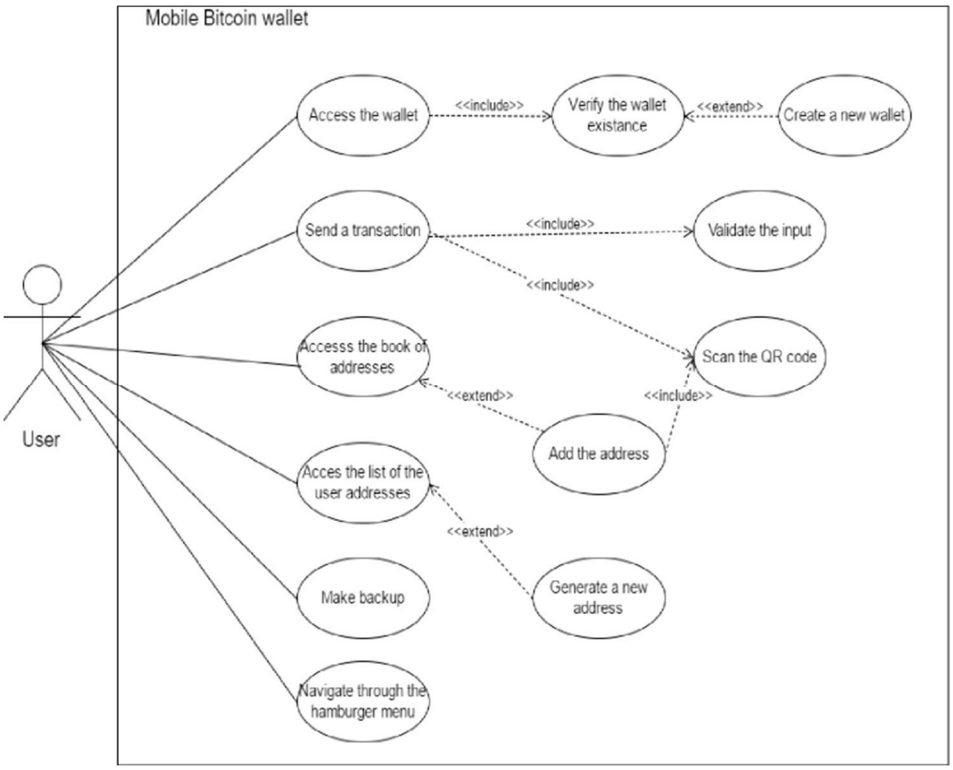

Figure 3. Use case diagram of the *Quackllet mobile Bitcoin wallet*

4.2. Project of Quackllet

The architecture of the *Quackllet* consists of layouts, activities, fragments as well as dialogues. All the components of an application are described in the *AndroidManifest.xml* file. Permissions, which are needed to access the protected parts of the API (Application Programming Interface) and permissions, which are required to interact with application components are also declared in this file. Manifest is generated automatically in Xamarin mobile application creation software and should not be modified manually.

The data layer of created mobile wallet consists of data elements, which will be processed and passed in the application. In case of the *Quackllet* application this data is a book of Bitcoin addresses as well as a *Bitcoin Secret*. The Bitcoin Secret is also known as WIF (*Wallet Import Format*) [15]. With Bitcoin Secret, private keys of the wallet can be retrieved. Thanks to this we can save and load our private keys, like in case of back up.

Both the book of addresses and Bitcoin Secrets are managed through the *DataManager*. This *DataManager* performs data serialization (book and Bitcoin Secrets) into an XML file, by the *XmlSerializer*. The serialized list of addresses from the book or secrets can be reloaded to the wallet by deserialization. The same as in case of serialization, *DataManager* is responsible for deserialization of XML files.

With this particular solution of data management, the application is able to have up-to-date information about contacts as well as the wallet owner addresses every time when the application is started. When the user is logging into his wallet, the XML file is being reloaded and the user is able to see the book of addresses consisting of addresses saved by him as well as his own addresses from the *AddressesFragment*.

The logic layer of the *Quackllet* contains information about the rules, which specify how the data can be created, stored or changed. In other words, the logic layer defines the processes or procedures, which are taking place in the Quackllet.

The first described process, which occurs at the first start of the wallet is the verification of the wallet owner. First of all, the Quackllet application checks whether there is already an existing user (not as the account, but if there was the wallet created before). Figure 4 depicts the piece of code responsible for verifying if there is an existing owner of the wallet.

```
startupWork.ContinueWith(t =>
{
    ISharedPreferences preferences = GetSharedPreferences(Constants.LoginInfo.LoginInfoPreferences, FileCreationMode.Private);
    bool existUser = preferences.GetBoolean(Constants.LoginInfo.HasLoginInfo, false);

    Intent intent;
    if (existUser)
    {
        intent = new Intent(this, typeof(LoginActivity));
    }
    else
    {
        intent = new Intent(this, typeof(FirstStartActivity));
    }

    StartActivity(intent);
}, TaskScheduler.FromCurrentSynchronizationContext());
```

Figure 4. Check up for an existing user of the wallet in *Quackllet*

Another procedure is the creation of the wallet during the first start of an application. The user has to enter the password with which he will be accessing the wallet. After that PBDKF (*Password-Based Key Derivation Function*) is initiated. It uses the salt, which is added to the password and the resulting text is being hashed and saved in the *SharedPreferences* together with the salt. This process assures the password security. Figure 5 presents the PBDKF procedure.

```
//generate the hash for the password
Rfc2898DeriveBytes pbkdf = new Rfc2898DeriveBytes(passwordText.Text, 32)
{
    IterationCount = 10000
};

//save the preference
ISharedPreferencesEditor editor = preferences.Edit();
//pbkdf
editor.PutString(Constants.LoginInfo.Salt, Convert.ToBase64String(pbkdf.Salt));
editor.PutString(Constants.LoginInfo.PasswordHash, Convert.ToBase64String(pbkdf.GetBytes(20)));
editor.PutBoolean(Constants.LoginInfo.HasLoginInfo, true);
editor.Commit();
Intent intent = new Intent(this, typeof(LoginActivity));
```

Figure 5. Securing the password with the PBDKF in *Quackllet*

The next process in logic layer is login into the wallet. It checks, whether the entered password, after PBDKF (so the hash is derived) is the same as the one saved in the *SharedPreferences* (hashes of salt and password combination are compared) (Fig. 6).

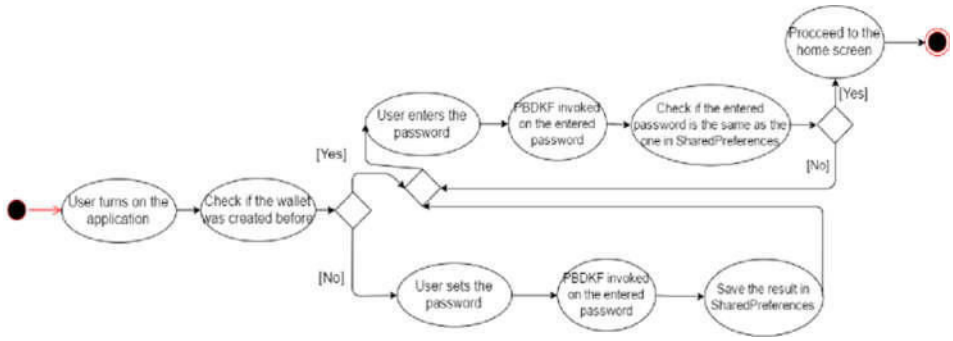

Figure 6. Activity diagram for login module of *Quackllet*

The addition operation is simply performed by adding a new address to the list of addresses. In the contacts option the user adds new contact by either scanning the QR code or by manual input. In case of the list of owner's addresses, a new address can be generated. While executing the send (Bitcoin) option the validation of the input data is performed. It checks if the fields for an amount, address and fee are not empty. Figure 7 presents this validation.

Furthermore, the user is able to choose how he would like to enter the address in the transaction process. He can either choose to scan the QR code, write it manually or add address from the contacts. This operation is performed by the *DialogFragment* in the send screen. Figure 8 presents the exemplary view of created application, presenting the send Bitcoin process.

All the functional requirements were fulfilled in the created mobile wallet. The user is provided with clear and easy to use user interface. For the security aspect the wallet is secured with the password. The password is encrypted using PBDKF and stored in SharedPreferences. Furthermore, the use of QR technology eliminates the possibility of entering wrong address in case of transaction. The backup option enables the user to

create the backup copy of the wallet and book of addresses. What is more, proper validation functions were made (e.g. input).

```
private bool validate()
{
    bool valid = true;
    if (TextUtils.IsEmpty(newTransaction.DestinationAddress))
    {
        addressText.Error = "Please enter address";
        valid = false;
    }
    else
    {
        addressText.Error = null;
    }
    if (newTransaction.Amount == 0)
    {
        amountText.Error = "Please enter an amount";
        valid = false;
    }
```

Figure 7. Validation of entered input in Bitcoin sending process in *Quackllet*

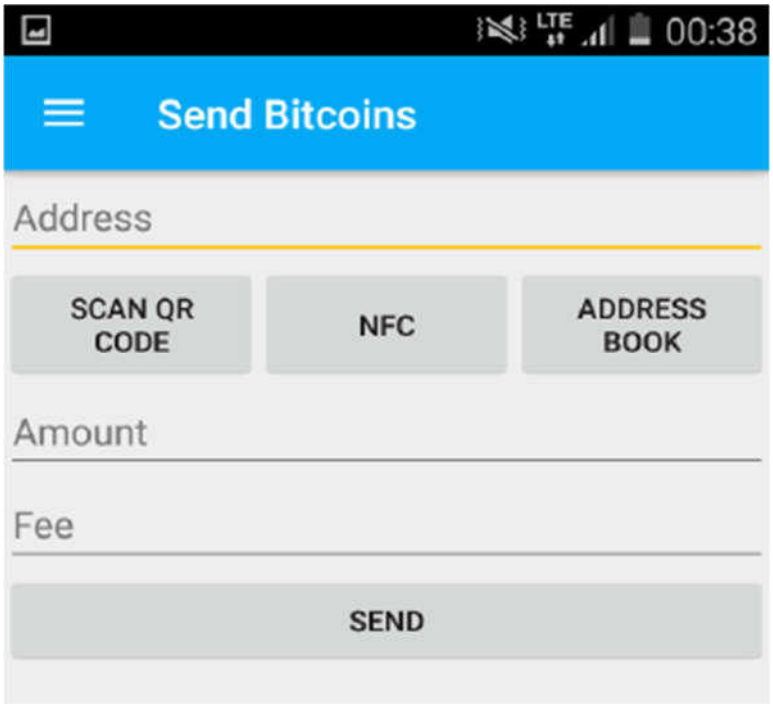

Figure 8. Send Bitcoins screen of *Quackllet*

5. Conclusions

The paper presented the security aspects of transactions in Bitcoin currency. It described the secure way of storing Bitcoins, because from the user point of view the security of transactions depends on ways of storing Bitcoins. The practical use of security solutions is the created mobile wallet. The security of the *Quackllet* application was fulfilled by creating such aspects as password encryption, validation of data or usage of QR code technology guaranteed the security aspect of the wallet.

The password encryption was performed using PBDKF. It not only hashes the password but also generates the salt (which is added to the password before hashing). The resulting hash is stored in the *SharedPreferences* and if the user wants to login to the Quackllet the hash of the entered password has to be the same as the one stored in the *SharedPreferences*.

The validation functions are made so the user would not leave the blank fields (e.g. in the sending screen the user cannot send the Bitcoins without entering the address as well as the amount of Bitcoins to send). What is more, the *Quackllet* application enables the user to make the offline backup of the wallet. It prevents from losing the Bitcoins and is easy to perform. The created backup file can be stored on memory card. The usage of QR code technology prevents from missclicking the entered address in case of transaction (if the user would make mistake in this step, he could send the Bitcoins to someone else).

The further development will be done to increase the security level of the created application and to improve its performance. Technologies such as NFC or a scanner of fingerprints would be developed in the future.

References

[1] M. Szymankiewicz, Bitcoin: Virtual currency of Internet (in Polish), Gliwice, 2014
[2] D. Homa, Secrets of Bitcoin and other cryptocurrency: how to change virtual money into real profits (in Polish), Gliwice, 2015
[3] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, http://bitcoin.org/bitcoin.pdf, 2008
[4] Bitcoin Foundation, https://bitcoinfoundation.org, 2017
[5] D. Ron, A. Shamir, Quantitative Analysis of the Full Bitcoin Transaction Graph, Proceedings of Financial Cryptography, 2013
[6] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J.A. Kroll, E.W. Felten, Research Perspectives and Challenges for Bitcoin and Cryptocurrencies, IEEE Symposium on Security and Privacy, 2015
[7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, Zerocash: Decentralized anonymous payments from Bitcoin, IEEE Symposium on Security and Privacy, IEEE, 2014
[8] E. Androulaki, G.O. Karame, M. Roeschlin, T. Scherer, S. Capkun, Evaluating User Privacy in Bitcoin, Proceedings of Financial Cryptography, 2013
[9] M. Babaio, S. Dobzinski, S. Oren, A. Zohar, On Bitcoin and Red Balloons, Proceedings of 13th ACM Conference on Electronic Commerce, pages 56-73, ACM, 2012
[10] J.A. Kroll, I.C. Davey, E.W. Felten, The Economics of Bitcoin Mining, or Bitcoin in the Presence of Adversaries, Proceedings of 12th Workshop on the Economics of Information Security (WEIS 2013), USA, 2013
[11] J. Becker, D. Breuker, T. Heide, J. Holler, H. Rauer, R. Bohme, Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency, Workshop on the Economics of Information Security, 2012
[12] I. Miers, C. Garman, M. Green, A.D. Rubin, Zerocoin: Anonymous Distributed E-Cash from Bitcoin, IEEE Symposium on Security and Privacy, 2013

[13] S. Barber, X. Boyen, E. Shi, E. Uzun, Bitter to Better – How to Make Bitcoin a Better Currency, Proceedings of Financial Cryptography, 2013
[14] G.O. Karame, E. Androulaki, S. Capkun, Double-spending fast payments in Bitcoin, Proceedings of 2012 ACM conference on Computer and Communications Security, pages 906-917, ACM, 2012
[15] NBitcoin library documentation: https://programmingblockchain.gitbooks.io/programming\\blockchain/content/introduction.Cited20Feb2017

Chapter 8

User-Perceived Performance Analysis of Single-Page Web Application Using Navigation, Resource and User Timing API

1. Introduction

Nowadays, it is not enough for a web application to perform some function. It must also look good, be comfortable and fast. It turns out that application performance is a critical factor, because if it is low, it frequently causes an average user to resign from the service, which translates directly into financial losses [1].

Such a state of affairs caused that the discipline concerned with research and facilitating web system efficiency has developed. It is concerned with all aspects of system operation, from processing queries by a server, through data transfer over the network, its protocols and mechanisms, to data processing and interface drawing by the user's browser (or other tool being the system client).

The assertion whether a given application is efficient or not is a matter of its direct speed of processing or data transfer. An efficient application is the one that allows users to perform tasks without the impression of delay. In an efficient application users are not surprised with a blank loading screen and well designed and realized interface will prevent their attention from being diverted from the page. Such a list of requirements concerning efficiency enables any user to create their own definition of an efficient system according to their own perception [2]. The authors – unable to research users' impressions – decided to identify efficiency as perceived by users with the speed of transfer and data measured in a browser.

In order to collect real measurements of efficiency as experienced by a user, it is necessary to perform Real User Monitoring, RUM. Real measurements, in contrast to the synthetic ones, show the actual efficiency, as experienced on user's computer, including all the relevant factors [3].

It turns out that synthetic and real measurements may indicate a completely different system status. Synthetic research confirms that a page is running and users are able to obtain the required content quickly and without errors. They can answer the question: "Does the system work fast?" but are unable to tell us whether the system works fast for everyone and always and on each device.

The authors aimed to research the influence of elements of a web system on efficiency as perceived by a user by means of performing real measurements (RUM). To

achieve it, they decided to use relatively new tools, yet gaining more and more popularity within the field, namely Navigation Timing, Resource Timing and User Timing API.

2. Related Work

The problem of web application performance is the focus of many research teams. In 2013, Panwar described the concept of Application Performance Management (APM). He demonstrated the significance of end to end application performance monitoring and management to achieve high accuracy in predicting application performance across multiple tiers e.g. databases, middleware network, etc. [4].

In the same year, Wert advocated the Performance Problem Diagnostics (PPD) approach which addresses the challenges of uncovering performance problems and identifying their root causes. He demonstrated that using operation monitoring data to derive variable workload models for systematic experimentation is a promising approach to uncover performance problems [5].

A large group of researchers focuses on delay measurements. Cito et al., investigated how performance engineers can identify three distinct classes of externally-visible performance problems (global delays, partial delays, periodic delays) from concrete traces. They developed a simulation model based on a taxonomy of root causes in server performance degradation. They extend their analysis and apply their methods to real-user monitoring (RUM) data [6].

In 2015, Karabin and Nowak analyzed the performance of Single Page Application (SPA) type web applications, constructed with the help of two JavaScript frameworks: AngularJS and Ember.js. They showed that the size of the framework or library is of no major significance for SPA type web applications, as the operations on the DOM tree and the speed of execution of JavaScript code are the actual bottlenecks [7].

In 2016, Stępniak and Nowak analyzed the effectiveness of methods accelerating the process of loading applications of SPA type, including the mechanisms offered by the HTTP/2 protocol. The results indicate that the technique accelerating the process of loading an SPA application the most is the minification of JavaScript code, which significantly reduces the size of transferred resources. The removal of unnecessary CSS rules, despite the small difference in reducing CSS files, undoubtedly has a positive impact on accelerating an SPA application [8].

3. Motivation and Choice of Experiment

User-side performance measurements are usually carried out exploiting the features provided by a web browser. They are accomplished by adding a special JavaScript code to the content of the page. This code is provided by the creator of the measurement tool. For the sake of the user's (system developer's) convenience, it is usually very short and is there only for authorizing and downloading the right measurement program.

Formerly tools operating in this way had limited operational possibilities. Web browsers did not make detailed information on webpage efficiency or speed of downloading resources available. Measurements tools had to inject JavaScript into

functions responsible for the measured system components or append own functions of servicing events handled by HTML elements (the best known example is the "load" event).

This state of affairs changed as a result of developing first RFCs, and then the popularization of dedicated interfaces and tools for JavaScript engines in browsers, supporting performance measurements. Those interfaces and tools are:

- Navigation Timing API – makes information concerning the course of process and speed of the basic process of loading a web page available.
- Resource Timing API – makes information relating to the course of processes and speed of loading of individual resources as loaded by the web page available.
- User Timing API – a set of functions making it possible to conveniently measure time between two (or more) points in the course of webpage performance [9, 10].

In technical literature much attention is paid to the description of the functions provided by the Navigation Timing, Resource Timing and User Timing interfaces. The authors, however, failed to find scientific publications presenting the results of genuine research with the use of those tools.

It can be seen that the profile of measurements carried out by the Navigation Timing and Resource Timing interfaces allows one to look at the performance of the systems from a different angle. Instead of determining the key performance determinants and only optimizing their values, they can be subject to measurements and then it can be researched how their performance depends on the respective stages of page loading (using the Navigation Timing API), resources (using the Resource Timing API), or other selected factors (using User Timing API). The aim of such research is to understand how the performance of a system depends on caching, overhead of network protocols, delays within networks and other investigated aspects covered by the interfaces discussed. The authors decided that due to a small number of similar analyzes, this type of research would be the most interesting.

4. General Description of the Experiment

The research was carried out on an existing, commercial web system. Taking into account the profile and cases of the system use, several key processes and critical time points were selected from the viewpoint of performance as perceived by the user. The measurements were implemented using Navigation Timing, Resource Timing and User Timing interfaces.

Because the selected tools operate within the browser, that is on the user's side, it was necessary to create a service that would collect data from the client devices and write them to the database for further analysis. In addition to measurement data, the system also collected selected information on the user's environment (browser, operating system), but without the data that would enable the identification or distinguishing between users (personal data, IP addresses).

The measurements taken are real measurements (RUM) as they were performed on the user's device and their results reflect the impact of all the conditions outside the

environment controlled by the system's creators. Incorporating measurement processes and key values for the selected system can be included in the custom metrics group.

The domain of the web system that was subject to experiment is the creation of an environment for integrating company employees. It is about maintaining good atmosphere within the organization of the company's customers by elements such as organizing joint integration meetings (social events), employees competing in different non-work related competitions, or mutual recognition of achievements and well-done work.

The system uses the following set of technologies:

- The entire application architecture is Single Page Application (SPA), which means that there is only one HTML document for the whole system, and the implementation of business logic is (largely) implemented on the browser's side in JavaScript. Recording and retrieving information from the server is accomplished by asynchronous HTTP requests. When using the system and navigating to other subpages, no reloading occurs.
- The server software was created utilizing a Ruby framework called "Ruby on Rails" using a relational database.
- Browser-based software was created using the JavaScript framework called "AngularJS". This information is important from the point of view of the implementation of measurements as AngularJS heavily interferes with the processing of an HTML document by a browser.
- Communication between the server and the browser software is performed by means of the HTTP/1.1 protocol.

Table 1. summarizes important non-technical system information.

Table 1. Important information about the researched system

Description	Data
Number of users	Approximately 17 thousand
Number of active users (last 60 days)	Approximately 6 thousand
Country of residence of the majority of users	United States
Users' profile	Employees of offices and large companies

5. Metrics

All measurements of each of the studied metrics are considered independently. It means that the measurements of two metrics do not affect each other, and for each one a separate record is saved in the database. Its structure is as follows:

- name of user's browser,
- version number of user's browser,
- a short marking of the platform the user is using (e.g. Windows, Mac),
- date and time of measurement,
- metric name,
- time – main metric value, e.g. page loading time.

User environment data is extracted from the "User Agent" header attached to each HTTP request performed by the browser. It should be borne in mind, though, that inference based on this header is not 100% reliable or effective as the user can easily

change its content, moreover rare browsers and platforms are not recognized. Such measurements must be omitted when developing analyzes that include the browser or the user's system.

Metric 1: Time to login page

When anonymous users first enter the system page, they encounter the login page. Bearing in mind that the first impression is the most important, it is a great disadvantage for the system, if already at the stage of the login form, the webpage seems to be sluggish.

In the investigated system, the login form is the only item that is outside the SPA architecture, which means that after entering the user name and password, the page is reloaded. It does not use any extra data retrieved from the server with asynchronous queries either. Only an HTML document is loaded, a Java script handling the login form, images, and CSS sheets. This is the only place in the entire system where one can rely on a web browser "load" event as the point at which the form is loaded and ready to use (this fact was supported by empirical research).

Due to the above facts, measurements of the first metric are very simple. They involve the implementation of the function evoked following the "load" event, which reads the measurements available through the Navigation Timing and Resource Timing interfaces, and then sends them to the data aggregation server.

The data recorded for this metric are:

- set of Navigation Timing API measurements,
- set of Resource Timing API Resource measurements for all resources of login page,
- main metric value: time from the start of page load to "load" event.

Metric 2: Time to first post

When a logged in user first enters the system page, they see the main panel of the system (the dashboard). This is where it is important that as soon as the user logs they see something relevant. The central part of this view is the list of posts, so the part of the system covered here is the process from the beginning of page loading (reloading after logging in) until the title of the first post is displayed.

After loading and processing the basic page document and attached resources, the system retrieves additional data from the server using asynchronous HTTP requests. The data include, among others, the list of posts to display.

When implementing this metric, the challenge is placing the measurement script as close as possible to the title appearance on the page. The best way to achieve this is to place it just below the HTML tag displaying the title of the post, the "<script>" including JavaScript instruction which perform measurements (or parallel solution taking into account the influence of JavaScript frameworks on HTML document processing by the browser).

The data recorded for this metric are:

- set of Navigation Timing API measurements,
- Resource Timing API measurement set for all panel resources that have been loaded until the first post title is loaded (usually before the "load" event when not all resources have yet been loaded).
- main metric value: time from the start of page load to the display of the first post.

Metric 3: Time to avatar

When a logged in user first enters the system page, they see the main panel of the system This is where it is important that as soon as the user logs he sees something relevant. Metric 2 (time to first post) treated the first post from the central list as a significant item. However, it is well known that not only the speed of showing significant information affects the perception of a website's performance. It is equally important for the user to simply see the progress of the loading process and the appearance of the subsequent portions of the interface. In addition to the list of posts, one of the first items that the user notices is their own avatar appearing on the main panel of the system.

After loading and processing the basic page document and attached resources, the system retrieves additional data from the server using asynchronous HTTP requests. This data includes, but is not limited to, the user's profile information and avatar.

When implementing this metric, the challenge is similar to the previous one, i.e. placing the measurement script as close as possible to the actual appearance of the avatar on the page. In this case, however, the best way to achieve this is not to use JavaScript instructions from an HTML document. Thanks to the image element properties in the HTML document, one's own JavaScript function can be registered, which will be executed when the image processing is completed. Images on a webpage, just like the page as a whole, generate a "load" event after processing. This allows the measurement script to be connected very close to the actual display of the image.

The data recorded for this metric are:

- set of Navigation Timing API measurements,
- Resource Timing API resource set for all panel resources that have been loaded until the first avatar is loaded (usually before the "load" event when not all resources have yet been loaded),
- in particular, the Resource Timing API measurements for the avatar image loading process,
- main metric value: time from the start of page load to the display of avatar.

Metric 4: Navigation time to event

If an organization (company) – whose member the logged user is – has social events recorded in the system (future or past), the user can go from the event list to the detailed view of a particular event. It is a very important and frequently performed action as events are one of the main components of the business model of the system.

During this navigation process the page is not reloaded. The whole navigation process begins when the user clicks on the event title and ends when the event description is displayed on the page.

The measurement of the total duration of this action is performed using the User Timing interface feature. At the moment of clicking on the event title on the list, the first time point is set, and when the event description has been loaded on the page, the second point is set. Then the lapse between the two points is calculated. Starting the measurement functions as close to the event description display as possible is realized in the same way as in the case of metric 2 (time to first post).

In addition to measuring the entire process with the User Timing API, asynchronous measurements of the resources loaded are also recorded. In this way, the system retrieves

the data necessary for navigation. The measurements are carried out using the Resource Timing API. In this case, Navigation Timing API measurements are not recorded because the webpage does not reload.

The data recorded for this metric are:

- Resource Timing API measurement set for all resources which have been loaded since the event title was clicked until the event description was displayed.
- main metric value: the time from the moment the event title is clicked until the event description is displayed.

Metric 5: Time to change participation in the event

If an organization (company) the logged user is a member of has a future social event recorded in the system, the user from the single event view level can request or cancel their participation in the event by clicking the "join" or "leave" button. One of the visual confirmations of the change is adding or removing the user's avatar thumbnail from the event attendee list. It is a very important and frequently performed action as events are one of the main components of the business model of the system.

During the process of changing the participation status in the event the page does not reload. The whole process starts when the user clicks the join/leave button and ends when the user's avatar appears or disappears from the list of participants.

The measurement of the total duration of this action is performed using the User Timing feature. When the join/leave button is clicked, the first time point is set, and the second point is set after adding or removing the avatar from the participants list. Then the lapse between the two points is calculated. Starting the measurement functions as close to avatar adding or removing as possible is realized in the same way as in the case of metric 2 (time to first post).

In addition to measuring the entire process with the User Timing API, asynchronous measurements of the resources loaded are also recorded. In this way, the system retrieves the data necessary to carry out the participation status change. The measurements are carried out using the Resource Timing API. Usually it is just one or two resources – sending a request to change the status to the server and possibly downloading the avatar. In this case, Navigation Timing API measurements are not recorded because the webpage does not reload.

The data recorded for this metric are:

- Resource Timing API measurement set for all the resources that have been loaded since the leave/join button was clicked until the avatar was displayed or removed,
- main metric value: the time from the moment of clicking the leave/join button until the avatar is displayed or removed.

6. Analysis of Research Results

6.1. Information and general statistics of measurement results

Table 2. gives an overview of the facts concerning the implementation of the conducted experiment.

Table 2. Information on the implementation of the measurements

Description	Data
Time of experiment implementation	2016-11-25 ÷ 2017-01-03
Number of measurements	16013
Number of measurements for metric 1	5490
Number of measurements for metric 2	4969
Number of measurements for metric 3	4864
Number of measurements for metric 4	429
Number of measurements for metric 5	261

The implementation details for loading and handling web pages, as well as the implementation of Navigation Timing, Resource Timing and User Timing measurements, vary depending on the web browser used. Table 3. shows browser usage statistics by users of the investigated system.

Table 3. User browser statistics

Browser	Number of measurements	Percentage of measurements
Google Chrome	10940	65%
Internet Explorer	4405	26%
Firefox	767	5%
Safari	435	3%
Microsoft Edge	202	1%
Opera	25	0%

The process of loading and operating a webpage in SPA is extremely complex. It is influenced by many factors that are most often difficult to predict and control. According to the authors the main reasons behind the disruptions during the experiment were:

- suspending JavaScript program performance caused by the fact that the user after activating a webpage begins to deal with other matters,
- visits of Web spiders (e.g. loading time of system login page did not exceed 100 milliseconds – in other cases it always exceeded 1 second),
- browsers (most often the older ones) did not properly realize measurement functions of research interfaces (negative time values).

These and other phenomena caused that the results of some measurements were distorted and should not be taken into account during the analysis. Hence 2540 measurement results (over 13% of the total) were verified negatively. The numbers and statistics presented previously relate to the measurements after discarding the incorrect results (16013 correct measurements).

6.2. Principles of analysis of measurement results

The aim of the analysis of the results of the experiment was to examine the relationships between the main values of the defined metrics and their respective components. To achieve this, a correlation study was performed. Pearson's correlation coefficient – determining the force of linear relationship between two variables and the Spearman correlation coefficient – determining the force of the monotonic relationship between the two variables [11,12] were deemed appropriate and sufficient by the authors.

Table 4. Factors affecting the main values of the metrics

Factor		Description
nav-fetch	Web page	Time from deciding that new page is to be fetched to HTTP request start. This includes asking browser cache, resolving DNS name and establishing TCP connection
nav-cache		Time of asking cache if requested page needs to be loaded from remote server or is present in cache
nav-dns		Time of resolving remote server DNS name
nav-tcp		Time of establishing TCP connection with remote server
nav-ttfb		Time from TCP connection established to first response byte received (time to first byte)
nav-dload		Time of downloading actual page document
nav-docLoad		Time of processing HTML document by browser
nav-resLoad		Time of processing additional resources requested by HTML document (browser already tries to handle these resources during processing of HTML document). This factor includes resources not complete during that time.
nav-fe		Time from receiving whole HTML document to page "onLoad" event (whole page processing in browser – frontend)
res-tcnt		Total count of resources requested by page
res-cac-cnt	Cached	Count of cached resources
res-cac-avg-dur, res-cac-med-dur		Average/median total time of resource loading and processing (duration)
res-ncac-cnt	Not cached resources	Count of not cached resources
res-ncac-avg-fetch, res-ncac-med-fetch		Average/median time from deciding that new resource is to be fetched to HTTP request start (like nav-fetch)
res-ncac-avg-dns, res-ncac-med-dns		Average/median time of resolving remote server DNS name (like nav-dns)
res-ncac-avg-tcp, res-ncac-med-tcp		Average/median time of establishing TCP connection with remote server (like nav-tcp)
res-ncac-avg-ttfb, res-ncac-med-ttfb		Average/median time from TCP connection established to first response byte received (like nav-ttfb)
res-ncac-avg-dload, res-ncac-med-dload		Average/median time of downloading actual resource (like nav-dload)
res-ncac-avg-dur, res-ncac-med-dur		Average/median total time of resource loading and processing (duration)
res-av-fetch, res-par-fetch	Avatar or Participation status	Time from deciding that new resource is to be fetched to HTTP request start (like nav-fetch)
res-av-dns, res-par-dns		Time of resolving remote server DNS name (like nav-dns)
res-av-tcp, res-par-tcp		Time of establishing TCP connection with remote server (like nav-tcp)
res-av-ttfb, res-par-ttfb		Time from TCP connection established to first response byte received (like nav-ttfb)
res-av-dload, res-par-dload		Time of downloading actual resource (like nav-dload)
res-av-dur, res-par-dur		Total resource loading and processing time (duration)

Data rarely reveal a strong linear relationship that can be examined using the Pearson correlation coefficient. It does not mean, however, that there is no dependence. Any monotonic relationship can be investigated using the Spearman correlation coefficient. It is good practice to compare both correlation coefficients for the same data.

It was decided that the interpretation of the correlation would be based on the coefficients calculated for all the collected measurements, regardless the division into

browsers. It means that the main part in their values was attributed to measurements collected from Google Chrome, which accounted for 65% of them. This decision was made to obtain a single value, on which the interpretation of the results for a typical average system user can be based.

6.3. Factors that affect the main values of the metrics

On the basis of the results of the collected measurements, 38 factors were selected on which the main values of each metric are potentially dependent. Table 4. presents an overview of each of the factors used while interpreting the results of the correlation analysis for each of the metrics.

All the described factors were calculated for each recorded metric session. That is why there are simple Navigation Timing values but average and median values for Resource Timing – single page load includes many resources which we would like to represent as a single number.

6.4. Analysis of measurement results of metric 1

Figure 1. shows Pearson and Spearman correlation coefficients for metric 1. The main value of metric 1 depends primarily on the process of loading a web page from scratch and processing an HTML document – relatively few additional resources are loaded.

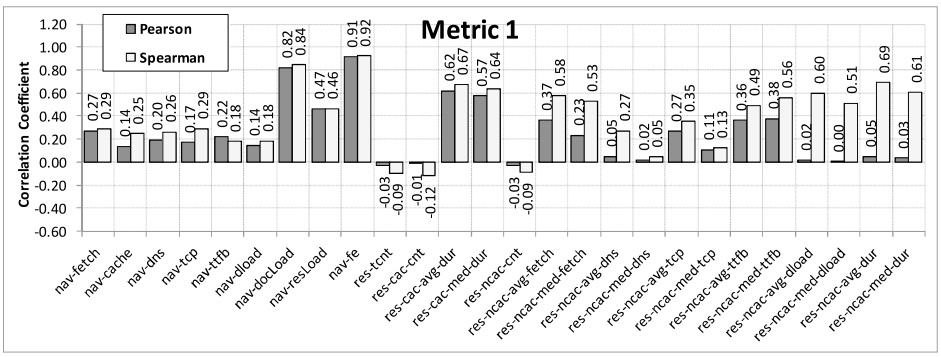

Figure 1. The values of Pearson and Spearman correlation coefficients for metric 1

The highest correlation coefficients are between the main value and the "nav-fe" and "nav-docLoad" factors. Those factors denote the total processing time of the whole page on the browser's side, and in particular the processing of an HTML document. The value of the correlation coefficient within the range of 0.8 ÷ 0.95 suggests very strong correlation.

A few other factors are in the mean correlation with the main metric value. They are associated with loading additional page resources. In this case, the Pearson coefficients are significantly lower than the Spearman coefficients. It means that these relationships are monotonic but not linear.

A very small impact on the main metric value of the overhead of protocols and network mechanisms such as DNS ("nav-dns", "res-ncac-avg-dns", "res-ncac-med-dns" and TCP ("nav-tcp", "res-ncac-avg-tcp", "res-ncac-med-tcp") is worth-noticing.

6.5. Analysis of measurement results of metric 2

Figure 2. shows the values of Pearson and Spearman correlation coefficients for metric 2. The main value of metric 2 depends primarily on the process of loading a webpage from scratch and processing a HTML document, but unlike metric 1, a lot of additional resources in the form of images, JavaScript, CSS or data are loaded (loaded from the server asynchronously).

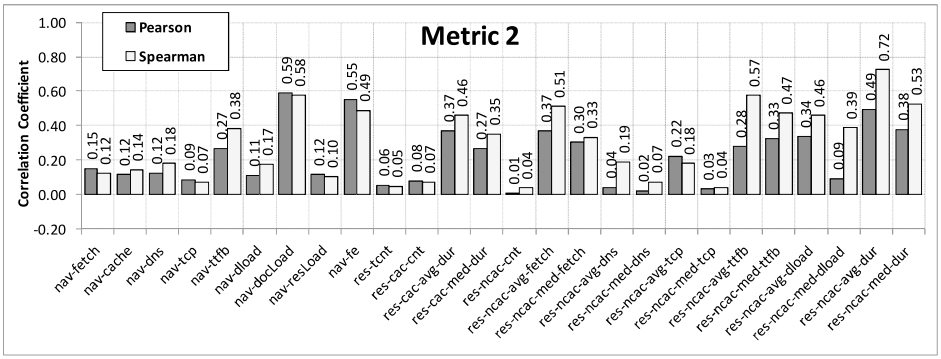

Figure 2. The values of Pearson and Spearman correlation coefficients for metric 2

The highest correlation coefficients exist for several factors. As with metric 1, there is a large dependency of the main value from the page processing on frontend ("nav-fe" and "nav-docLoad" factors), also with an emphasis on the processing of a HTML document. This is not the dominant relationship, however.

Equally high coefficients can be seen for the values of average load times of a single resource. They are symbolized by "res-ncac-avg-dur" and "res-ncac-med-dur" factors. Also, the significant dependence on the "res-ncac-avg-ttfb" factor, which includes, among others, the average processing time of HTTP requests by the server, can be seen.

The significant effect of the "res-ncac-avg-fetch" factor is noticeable. This factor indicates the time from the beginning of the resource request to the start of the HTTP request, thus including a TCP connection and resolving the DNS address. With more resources loaded, the main value is increasingly more dependent on the overhead of the network mechanisms.

The higher values of Spearman coefficients over the Pearson ones mean that the investigated relationships are monotonic, but not necessarily linear.

6.6. Analysis of measurement results of metric 3

Figure 3. shows the values of Pearson and Spearman correlation coefficients for metric 3. This metric is very similar to metric 2. Its main value depends primarily on the process of loading a webpage from scratch and processing an HTML document, but

unlike metric 1, a lot of additional resources in the form of images, JavaScript, CSS or data are loaded (loaded asynchronously from the server).

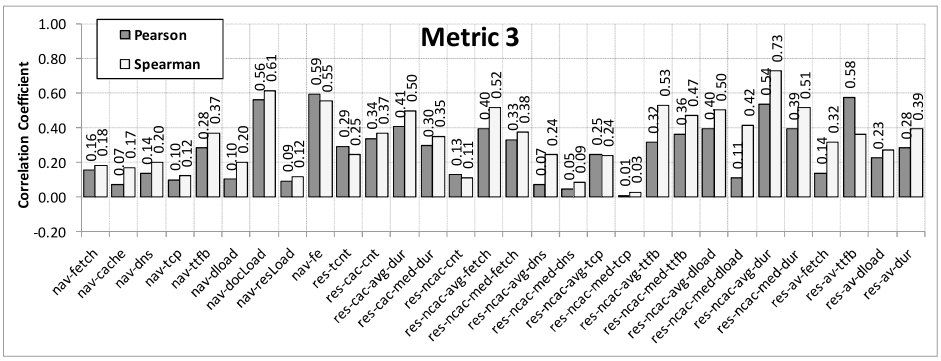

Figure 3. The values of Pearson and Spearman correlation coefficients for metric 3

The highest correlation coefficients exist for several factors. As with metric 1, there is a large dependency of the main value from the page processing on frontend ("nav-fe" and "nav-docLoad" factors), also with an emphasis on the processing of a HTML document. This is not the dominant relationship, however.

Equally high coefficients can be seen for the values of average load times of a single resource. They are symbolized by the factors "res-ncac-avg-dur" and "res-ncac-med-dur". Also, the significant dependence on the "res-ncac-avg-ttfb" factor, which includes, among other issues, the average processing time of HTTP requests by the server, is visible.

The significant effect of the "res-ncac-avg-fetch" factor is noticeable. This factor is responsible for the time from the beginning of the resource request to the start of the HTTP request, and therefore includes a TCP connection and a DNS address resolution. It means that with more resources loaded, the main value is increasingly more dependent on the overhead of the network mechanisms and protocols.

As the main value of this metric depends on when the user's avatar is drawn on the screen, the relationship between that value and the process of loading the resource of that avatar has also been investigated. The value of the coefficients shows that this relationship is statistically significant, but relatively weak compared to the aforementioned factors.

6.7. Analysis of measurement results of metric 4

Figure 4. shows the values of Pearson and Spearman correlation coefficients for metric 4. The main value of the metric no longer depends on the process of loading a web page from scratch and processing a HTML document, as is the case with metrics 1, 2 and 3. The main value of the metric is based on resource loading.

The highest correlation coefficients are found between the main value and the "res-dur" and "res-cac-avg-dur" factors, i.e. the average loading times of a single resource stored in the cache. It is clearly evident that only Pearson's correlation coefficients are very high, while Spearman's coefficients show a very weak correlation. Such a situation means that the measurement data are somehow distorted by outliers cases. The Pearson's correlation coefficient is prone to such phenomena. In such a case, the Spearman

correlation coefficient should be used, which is resistant to similar phenomena. For this reason, these two values cannot be considered correct, and the interpretation of the results for these factors should be based only on the Spearman correlation coefficients.

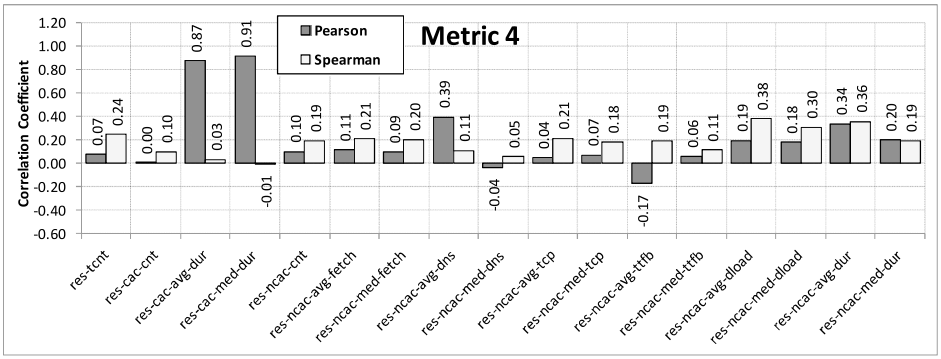

Figure 4. The values of Pearson and Spearman correlation coefficients for metric 4

Among the other factors none show a strong or moderate correlation with the metric's main value. Only the "res-ncac-avg-dur" factor stands out slightly, but the correlation coefficient 0.34 is on the border of significance. This situation may be caused by two phenomena:

- the main metric value depends evenly on many factors,
- factors that are not included in Resource Timing, such as interface drawing time or JavaScript resource processing, influence performance to a large extent.

6.8. Analysis of measurement results of metric 5

Figure 5.8 shows the values of Pearson and Spearman correlation coefficients for metric 5. The main metric value does not depend on the process of loading a web page, but is based on loading a small number of resources, specifying one master resource.

The highest correlation coefficients were obtained for "res-ncac-med-dur" and "res-ncac-avg-dur" factors. This means that for metric 5, the primary performance element is the average time spent loading resources. The Pearson correlation coefficient above 0.9 indicates a very strong linear relationship.

Slightly lower correlation coefficients were obtained for "res-par-dur" and "res-par-ttfb" factors, i.e., the waiting time for the first byte of the master resource response for metric 5 (communication with the server to change the attendance status of the event). It is an expected phenomenon as the action constituting the base of metric 5 most often sends only one asynchronous HTTP request.

The pairs of correlation coefficients for the factors "res-ncac-avg-ttfb" and "res-ncac-med-ttfb", corresponding to the average waiting times for the first byte of the HTTP request response are extremely interesting. Spearman correlation coefficients are very high, while Pearson's correlation coefficients are close to 0. This situation means that there is a strong dependency of the main metric 5 value on these factors and that it is monotonous, and definitely not linear.

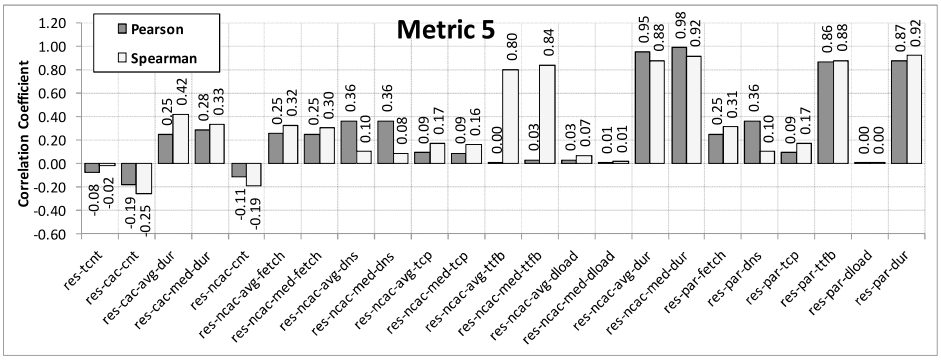

Figure 5. The values of Pearson and Spearman correlation coefficients for metric 5

6.9. Summary of results

At the top level the metrics can be divided into those that cover the loading process from the web page's base and those that do not.

The first group includes metrics 1, 2 and 3. In their case, the most significant influence on the main values were the factors that meant aggregate page processing times by the browser. They are responsible for most of the duration of the page loading process. In the process of processing a webpage on the client's side of the system, the stage of processing an HTML document and the processing of resources to which the references in this document are embedded can be distinguished. Of these two stages, the first step, i.e. the processing of an HTML document, has a greater impact on the metric's main value.

Metrics 2 and 3, unlike metric 1, include a large number of additional resources such as user avatars and server communication to retrieve information from asynchronous HTTP requests from the database. For these two metrics, the mean factor for the average aggregate load time of a single resource is also significant.

Metrics 4 and 5 belong to a group that does not involve loading a web page, but merely an exchange of information with the server to retrieve new data and, if necessary, download additional resources, such as images. In this situation, the primary factor influencing performance is the average aggregate load time of a single resource. The resource loading process is divided into smaller steps, the most significant of which is the time to first byte waiting time, including the server's request processing time. However, this is not a factor that dominates over others decisively.

While in the case of metric 5 the communication with the server and resource loading are a major factor in time, for metric 4, resource loading has an average or low impact on performance. This is due to the long processing time of loaded resources through the browser and JavaScript. The resource processing process is included in the main metric value, but is not covered by Resource Timing and Navigation Timing.

7. Conclusions and Directions for Further Work

The TCP overhead and DNS network mechanism have little impact on the performance of the web system under investigation. This conclusion was made based on the analysis of the results for "nav-dns", "res-ncac-dns-dns", "res-ncac-med-dns" (DNS) and "nav-tcp", "res-ncac-avg-tcp", "res-ncac-med-tcp" (TCP) factors. The fewer resources a function of the system includes, the smaller the impact is. It is justified as a smaller number HTTP requests means less need to use these process elements. Also, when loading resources, the dependence of the main value of metrics on the factors responsible for the measurement of these overheads is poor compared to other factors.

In the process of loading a web page, the dominant impact on performance is the processing of the page and its resources after they are downloaded, on the client side of the system, in the browser. HTTP server processing time is far less significant. This conclusion corresponds to the expectations as in systems built in the Single Page Application architecture most of the system logic is implemented on the client's side. This conclusion is based on the analysis of the results for "nav-fe" and "nav-docLoad" for metrics 1, 2 and 3. In each case, the correlation of these factors with the metric's main value is very strong.

Where the function of the system is based solely on loading and processing resources and asynchronously downloaded server data, the dominant impact on performance is the total time taken to retrieve a resource. This conclusion was based on the analysis of the results for the "res-ncac-avg-dur" and "res ncac med dur" factors for metrics 4 and 5.

In a situation when a system function is performed only by asynchronous communication with the server and retrieving additional resources, the time of processing HTTP requests by the server and transferring data to the user increases. It becomes the most important factor next to the total loading time of the entire resource (conclusion 3). The conclusion was based on the analysis of the results for "res-ncac-avg-ttfb" and "res-ncac-med-ttfb" factors, suggesting the mean times of "Time to first byte" (TTFB) values. The elements that make up the TTFB value are the time of processing the query by the server and the data transfer over the network.

Navigation Timing APIs and Resource Timing APIs are good tools for analyzing the impact of network protocols and processing and page loading processes on performance, but they do not provide a complete picture. This is confirmed by the case of metric 4, where a large part of the execution time is contained in the processing stage of the browser and JavaScript. It is necessary to use additional tools, such as User Timing API or custom measuring mechanisms.

The User Timing API allows one to investigate the execution time of any piece of JavaScript or execution of system functions if a situation permits placement of tags in appropriate places. In this experiment, this tool was used very generally only to measure the main values of metrics 4 and 5.

The results of the study allow us to unambiguously state which components of the system under investigation affect the perceived efficiency most. Definitely processing of the site and resources is the longest on the browser side and to this part of the system developers should devote the most attention. Optimizing the performance of server part, while also very important, has less impact on performance than processing data in the

browser. Indicating specific fragments that negatively affect the speed of the frontend requires a separate detailed analysis for each application and system function studied.

Studies have shown a relatively small impact of network overruns and protocols on performance. It turned out that the greatest impact on performance is exerted by the speed of processing the page and resources on the browser side of the user. Relatively small but noticeable was the impact of HTTP server processing time and data transfer time. It was shown that performance analysis using Navigation Timing and Resource Timing as the main tools is not complete and needs to be supplemented by the advanced use of the User Timing API, as it does not include with its scope the data processing stages through JavaScript and the processes responsible for drawing the on-screen interface. An appropriately designed study will identify the least efficient elements of the program. However, these results will be valid only for the system under study.

It would be worth-considering to carry out an analysis of threats to external validity by means of repeating the research for other SPA application. In the case of re-determining Pearson and Spearman coefficients, because statistical tests have been performed several times and are prone to a problem of multiple hypothesis tests therefore obtained results can be adjusted in some way (e.g. by Bonferroni correction) before their interpretation. It should be added that measurement data could be analyzed using PCA as an explanatory method.

The Web Performance Working Group constantly develops the Navigation Timing API and the Resource Timing API. Recommendations for the next levels of these standards are planned for 2017 and are to introduce a number of new measurement capabilities and improvements to the current functionalities. It will probably take another year or two before browsers adopt these new standards in their products. Studies similar to these but using these new interfaces may allow for even more detailed analysis and further conclusions.

Currently the standard on the Internet is HTTP/1.1, but HTTP/2 is increasingly used. The new version introduces a number of changes that may cause the results presented here to be outdated. To complete them, similar investigations using HTTP/2 should be performed.

References

[1] Gomez, Inc., Why Web Performance Matters: Is Your Site Driving Customers Away?, Whitepaper, 2010, http://www.mcrinc.com/Documents/Newsletters/201110_why_web_performance_matters.pdf
[2] I. Molyneaux, The art of application performance testing, 2nd edition, O'Reilly Media, Inc., USA, 2014
[3] D. Sillars, High Performance Android Apps, O'Reilly Media, Inc., USA, 2015
[4] M. Panwar, Application Performance Management Emerging Trends, International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, IEEE, 2013
[5] A. Wert, Performance problem diagnostics by systematic experimentation, 18th international doctoral symposium on Components and architecture (2013), 1–6
[6] J. Cito, D. Gotowka, P. Leitner, R. Pelette, D. Suljoti, S. Dustdar, Identifying web performance degradations through synthetic and real-user monitoring, Journal of Web Engineering, Vol. 14, Issue 5-6, (2015), 414–442
[7] D. Karabin, Z. Nowak, AngularJS vs. Ember.js – performance analysis frameworks for SPA Web applications (in Polish), Od procesów do oprogramowania: badania i praktyka, PTI, (2015), 137–152
[8] W. Stępniak, Z. Nowak, Performance Analysis of SPA Web Systems, 37th International Conference on Information Systems Architecture and Technology, Pt. 1, Springer (2017), 235–247

[9] B. Bermes, Lean Websites, SitePoint, Australia, 2015.
[10] P. Meenan, How Fast is Your Website?, Communications of the ACM, Vol. 56, No. 4 (2013), 49–55
[11] H. Garner, Clojure for Data Science, Packt Publishing, USA, 2015
[12] J. Hauke, T. Kossowski, Comparison of Values of Pearson's and Spearman's Correlation Coefficients on the Same Sets of Data, Quaestiones Geographicae, 30(2), 2011

Chapter 9

Real-time Control of the Mitsubishi RV-2F Robot

1. Introduction

The main purpose of this paper is to demonstrate how flexible are today's computers and how huge possibilities are in our reach. The authors show that real-time image processing to control the robotic arm is possible even in the case of a very inexpensive hardware which is the Raspberry PI v2.0 microcomputer.

The work begins with a description of the robot arm used during the experiment. It is a fully functional industrial robot that can be controlled via Ethernet. Next comes a closer look at the control system.

The control system design is the main subject of section three which depicts all main features of it. Features like an operating system under which the control of the Raspberry PI is working, the specification of task types used during the processing and some basic design assumptions.

In the succeeding part of the paper the vision system is described. The basic information about the cameras configuration and image processing done during experiments are shown.

The paper end is devoted to the time analysis of the specific parts of the control system. Jitter of all control task types execution time is presented as well as the summary of timing measurements.

2. General characteristics of the Mitsubishi RV-2F-D industrial robot

The smallest and lightest industrial robot manufactured by the Mitsubishi company is denoted RV-2F-D, see [4]. It belongs to the vertical multi joints robot type. The following parameters characterize the robot: 6 degrees of freedom – motion of six joints J1, …, J6, the AC servo motor drive system (the joints J2, J3 and J5 are equipped with the brake), the position repeatability ±0.02 mm, the absolute encoder position detection method, the maximum reach radius 504 mm, the maximum load capacity 3 kg, the robot's own weight 19 kg. Apart from the mentioned the following parameters shown in Table 1 are the most significant to the robot operation.

Table 1. Summary of robot parameters

Joints	J1	J2	J3	J4	J5	J6
Operating range	±240 °	±120 °	0 to 160 °	±200 °	±120 °	±360 °
Maximum speed	300 °/sec	150 °/sec	300 °/sec	450 °/sec	450 °/sec	720 °/sec
Maximum transposition during single control cycle	2,13 °/7,1ms	1,06 °/7,1ms	2,13 °/7,1ms	3,2 °/7,1ms	3,2 °/7,1ms	5,11 °/7,1ms
Tolerable moment				4.17 Nm	4.17 Nm	2.45 Nm
Tolerable amount of inertia				0.18 kgm^2	0.18 kgm^2	0.04 kgm^2

Figure 1. At the left Mitsubishi RV-2F-D industrial robot is shown. At the right the interior of one of the robot arms is shown. It is the arm between the J2 and J3 joints. There are two motors, two gears, two tensioners and two toothed belts. They constitute the two-joint drive

Figure 1 shows the robot. The robot software enables to accelerate or decelerate automatically the robot arms. This automation is based on the following parameters: the robot operational position, robot posture and a current load (a workpiece can be present or not). In this way the robot can operate at the best speed level dedicated to each performed task. The mechanisms and automates in question are introduced by the manufacturer and do not require an intervention of a user. The cycle time is reduced. The cycle time is defined as the time required for a back-and-forth movement over a vertical distance of 25 mm and horizontal distance of 300 mm when the load is 1 kg. In the case of the RV-2F-D Mitsubishi robot the cycle lasts 0.6 sec.

3. Control system design

The described system consists of 4 basic elements: the Mitsubishi RV-2F-D industrial robot arm, two PS3 eye cameras and Raspberry PI microcomputer, see [5]. The schema of connections between each pair of components is shown in Figure 2.

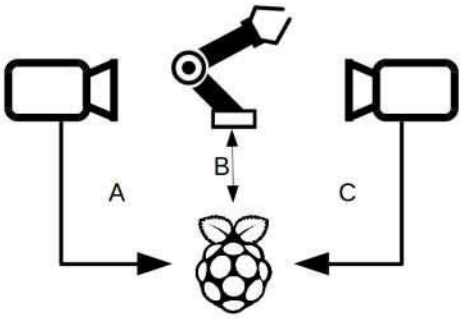

Figure 2. Control system simplified diagram. Arrows labeled as A and C symbolize the USB connections, arrow labeled as B symbolizes the Ethernet link

The Raspberry PI was loaded with the latest available Raspbian Jessie (release ID) distribution with the custom Linux kernel image. The kernel was patched Linux 4.4.50 kernel downloaded from the Raspberry repository, which is a fork of official kernel. The patch applied on the kernel was RT preempt in the 4.4.50-rt63 version, available in [6]. The RT preempt is commonly used to change the regular Linux kernel into one which can be used in Real Time applications.

To fully saturate the four ARM cores of Raspberry PI the control application was a single RT process, the multi thread application. All threads activities: the creation, data synchronization and joining, were done with the usage of the latest the C++14 standard implementation. The C++ standard library delivers the std::thread name space which aggregates the convenient wrapping utilities on the well-known POSIX thread library - pthread. The processing was split into three basic task types:

1. Camera image processing – two instances
2. Monitoring communication with robot – one instance
3. Control communication with a robot and data synchronization – one instance

The first type task performs only image processing operations and delivers the processed data to the control task. The task of the second type is responsible only for gathering the monitoring data from the robot, such as actual feedback from the robot encoders and AC motors current. This is mostly used for logging purposes and has the lowest priority. The third type of task is devoted to sending and receiving the set point acknowledges from the robot, this task is also receiving the data from the camera image processing tasks and performing simple algorithm to validate that the robot can perform the movement without problems. More about the visual transformations is given in section 4.

To calculate the control action duty cycle duration of the Raspberry PI application the theoretical duty cycle of the tasks devoted to robot communication was taken under consideration. The Mitsubishi's monitor communication manual claims that the Ethernet communication duty cycle is about 7.1 ms which depends on the internal robot configuration and can by modify in limited way only by authorized service. This time is not fixed and may vary because of the delays on the Ethernet link which should be deniable in the experimental setup where the network was built of two elements only, the robot was directly connected to the microcomputer. This led the authors to a conclusion

that there is no point in generating the control more often than once every 7.1 ms. This value is used as the reference duty cycle duration for the visual transformation.

As the cameras in the system two PlayStation 3 Eye are used. These devices are able to generate the color 320x240 video stream with a high frame rate. On the top performance the duty cycle of the frame generation can be shorter than 7.1 ms. The cameras are connected via the USB bus to the microcomputer which is equipped with the 4 USB A type ports. The 100 fps rate is used in the control implementation, as the one which is possible to handle for Raspberry PI and is the closest to the Ethernet communication duty cycle. The details of the image processing done in this task type are shown in section 4.

As the input to the control algorithm three parameters are consider: the cameras input images transformed by the edge detectors, the possible maximal transposition of the robot effector during one duty cycle - the result of robot parameters, and reference path to the target position. Based on those three parameters the control can be calculated. The cameras input are used as additional information, the control algorithm may analyze the part of the image where the effector will be in next 7.1 ms, in the matter of background shape changes, and can stop or choose the other path to continue the movement and bypass the obstacle and avoid a collision. In this paper the simpler version of the algorithm is considered, only the stop action is possible to be performed.

The time stability analysis is given in section 6.

4. Vision system description

As it was mentioned before the system described in this paper contains two PlayStation 3 Eye cameras. Those devices are well known as the fast and robust, and are often chosen as the IR detectors due to the easy lens filter installation. The lens of PS3 Eye is able to work in two modes: the close-up and 75^{o} field view. In this setup the field view option was exclusively used.

The cameras are able to generate a stable 125 fps data stream, equivalent to the 8 ms duty cycle, with the 320x240 resolution in the YUYV format. In addition they are compatible with the V4L2 drivers which simplifies their usage. The frame rate may be increased up to 187 fps but the resolution need to be decreased to 160x120. In other way the data stream would be larger than the USB 2.0 bus capacity.

Both cameras have been always configured in the same way to avoid an additional scaling which would be necessary in a case of the divergent configuration. During the preparation phase two basic types of cameras settings were tested:

1. Resolution: 320x240 with 50 FPS (20ms duty cycle).
2. Resolution: 160x120 with 100 FPS (10ms duty cycle).

Experiments show that despite the fact that the cameras are able to deliver much higher resolution at the chosen frame rates Raspberry PI is not able to process such amount of data in the desired time. As the final setup the second configuration was used.

The frame processing is done in two steps, the same for both cameras, firstly the Sobel's operator is applied on the currently received frame. The result of this operation is shown in Figure 3. After that the subtraction is performed on the current and previously

processed frame to get the points which have changed their location. That can allow to detect the external obstacle or detect the robot movement.

The obstacles detection may be enhanced in the heuristic movement prediction. The control algorithm has the information about the path and the arm set points, so it can decide which part of the image has to be processed and which can be skipped to get enough data to verify the future position of the arm. Such optimization should improve the performance and the control duty cycle time stability.

All image processing is done with the usage of the OpenCV library which is fully available for the Raspbian distribution. It was mentioned earlier that PS3 Eye cameras are compatible with the V4L2 library, this feature allows to integrate those devices with OpenCV library because V4L2 is natively supported by the framework.

The OpenCV framework is flexible tool which delivers such complicated features as: automated getting the frame from the camera, applying visual transformations on the received frame, and delivering the image in a simple format for the future processing. Among many the edge detection algorithms are also delivered inside the framework. The Sobel's and Scharr's operators are two which were consider to be used in the paper. During the time stability measurements the Scharr's operator shown the tendency to be the faster one. The time needed to transform the frame by the Scharr's operator was on the average 4-5% shorter than the time which was needed by Sobel's to perform the same transformation, but the edges found by it were significantly more blurry, which is unacceptable in the noisy robot environment. The Sobel's operator was chosen as the more reliable.

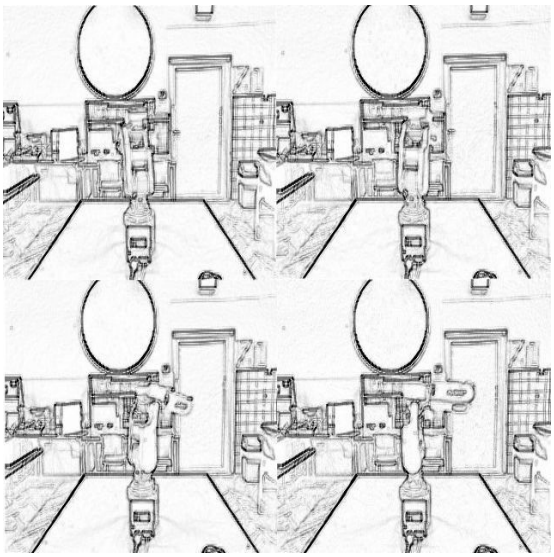

Figure 3. Collage of the vision from the cameras after Sobel's transformations. The figure is printed in the inverted color scheme

In figure 3 four frames of the film after the Sobel's transformation are shown. They seem to resemble a game called "Search for differences in drawings". The frames actually

show a still background and a moving robot. The background consists of many details. Among others we can notice: a door, a wall covered with tiles, and a round ring (to fight Sumo robots) hanging on the wall. The Mitsubishi robot located on the table is moving. Detection of this movement is visible by comparing the edge changes on individual drawings.

Likewise the robot motion, it is possible to detect static and moving obstacles using the camera and the Raspberry PI microcomputer.

5. Real-time trajectories executed for different robot powers

Our considerations related to a real-time control of the robot we start from a simple experiment conducted with the robot. We prepare a program to move the endpoint of the robot between several points. We try to activate six joints or nearly all of six joints. We will repeat the same action for two different levels of the power, i.e. 10% and 100 %. The results of these experiments are depicted in Figure 4. All robots positions are measured by the internal robot controller and send every 7.1 ms to the Raspberry PI computer where the monitoring task is logging them.

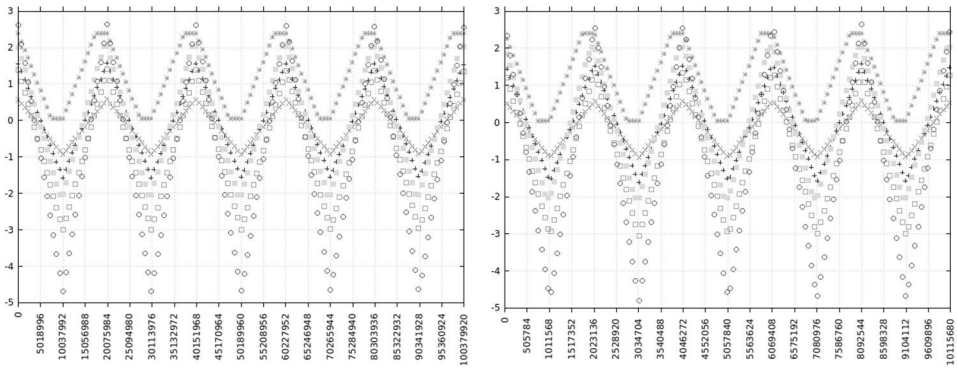

Figure 4. Trajectory comparison during robot autonomous control. At the left is shown the 10% speed override. At the right is shown 100% speed override. On the X axis is the time in µs. On the Y axis is the joint position in radians. Joint trajectories are marked in the following way: asterisk, plus, square, filled square, circle

We are surprised that the left and right picture trajectories are nearly the same. The robot moves from point to point. The coordinates of these points are identical for each of the trajectories (10% and 100% of power). In [2] different trajectories planning strategies were presented. It looks that a default producer configuration uses basic linear point interpolation without taking care of simultaneously finishing movement of all joints.

Nevertheless, each trajectory is characterized by a different dynamic. Moreover, at different speeds, not the same trajectory points are collected and stored in the computer memory. This makes it difficult to compare the results. On the other hand, the manufacturer's parameters for the robot shown in Table 1 dictate that we are dealing with a device with a very high accuracy and repeatability.

Have our expectations for bad results been confirmed, or does our interpretation of results fail? The same observation correspond to the controller.

In fact, the answer to these questions is difficult. The authors asked at the source, i.e. the Japanese robot maker. Unfortunately, they did not receive sufficient information. Well, in principle, the manufacturer is supposed to ensure the correct operation of the robot when performing typical robotic tasks. However, detailed signal nuances are often inaccessible to the customer as confidential data. How can a researcher and an experimenter deduce how angles and moments in robot joints actually behave in real time? This is a question about tools for measuring and processing signal data. Maybe it's also a question of the research courage. The monitored time stability measurements results are presented in section 6.

6. Time stability analysis

The main concern during creation of a control system on such tiny computers as Raspberry PI is the time stability of a generated control. Figures 5 and 6 show the jitter histograms of the image processing, robot control and communication tasks.

In Figure 5 it can be noticed that for the smaller image frames, tasks are performing much better in terms of the time stability than during processing bigger frames. It is noticeable that for the frame size 160x120 the most cycles are in the range 9,75-10,25 ms, where for the frame size 320x240 the significant amount of duty cycles are not in the point.

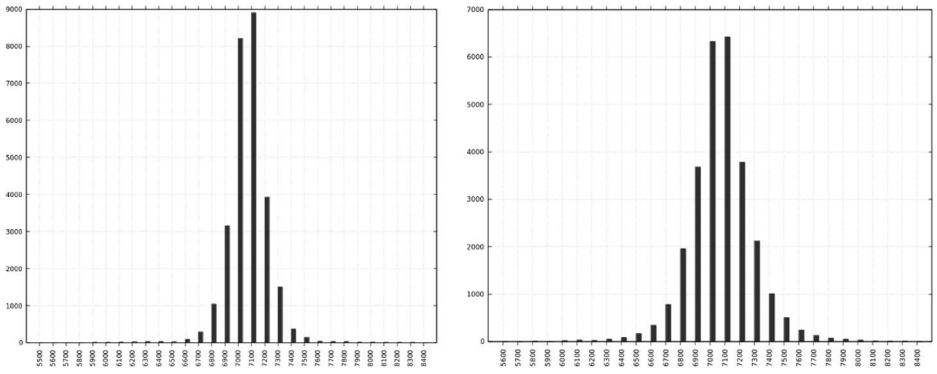

Figure 5. Time analysis of threads responsible for robot communication. At the left the control thread is shown. At the right the monitoring thread is shown. On the X axis is the control cycle duration in µs. On the Y axis is the number of samples

In Table 2 the summary of the time analysis for all type of robot control process tasks is illustrated. All measurements are performed for two types of camera input and for two different process priorities: the regular Linux process priority and the real-time process priority (RT).

It can be noticed that the maximal cycle time for all configurations occurs to be much higher than the expected. As it was mentioned in section 4 the setup with cameras

configurations for the 320x240 resolution was very unstable. It happens that even for the process RT priority some of the duty cycles takes more than 125 ms, this is almost 18 cycles of the robot communication. During this missed time the arm could collide with the unexpected obstacle or move in incontrollable matter, which under the load could result in a hardware damage. Much safer setup with the camera resolution equal to 160x120 also had troubles with the time stability. The maximal duty cycle in this case was about 62 ms which is almost equal to 9 robot communications cycles. In addition the connection between the processing of the bigger frame and the control thread unpunctuality can be observed. When bigger frames are transformed the processor tends to delay always other tasks which are not even on the same core. This indicates that the OS has no more resources to perform its own tasks and the random threads are interrupted.

The results of time stability measurement may be compared with [3] where the different OS than RT Windows was used.

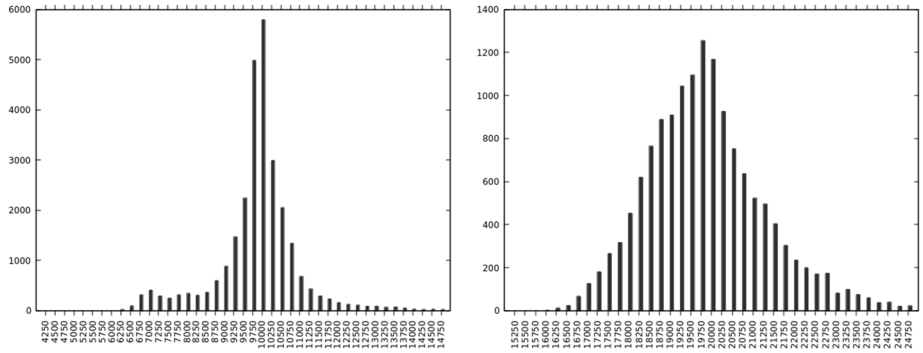

Figure 6. Time analysis of threads responsible for the camera communication and vision transformations. On the X axis is the control cycle duration in µs is shown. On the Y axis is the number of samples is shown

The real-time quality is measured by a jitter (see for example [1]). So it is no surprise that much attention was paid to the jitter study in this work. Figures 5 and 6 depict that the time stability is worth measuring because based on the result it is possible to choose safer/better solution.

Table 2. Summary of time accuracy. All results are shown in µs

Measure type	Task priority	Monitor		Command		Camera	
		160x120	320x240	160x120	320x240	160x120	320x240
Min cycle time	RT	2793	2790	3748	3251	6346	16196
	normal	2788	2780	3505	3780	6339	14757
Max cycle time	RT	59046	123477	60180	127974	62964	135641
	normal	189280	189785	190811	191976	190132	194684

7. Summary

The object of interest of the paper is the Mitsubishi industrial robot, and in particular its accurate and repetitive operation. The robot control system measures the angular positions of all six robot joints. These measurements are accurate and repeatable for different speed of the robot trajectories (see Figure 4). The authors, impressed by the operating precision of the device, decided to use an external measuring tool completely independent of the robot's manufacturer. This tool was the camera and microcomputer Raspberry PI. To what extent is this tool working in real time is shown in Figures 5 and 6 and in Table 2

As expected, a cheap microcomputer such as Raspberry PI cannot be used as a hard real-time system, but can be used as a soft RT system. What has been shown. The Raspberry PI microcomputer, despite its limitations, allows surveillance and obstacle detection even in such complex systems as the Mitsubishi robot.

References

[1] K. Kolek, A. Turnau, FPGA as a part of MS Windows control environment, Computer Science, vol. 8, no. 3, 2013, 61-68
[2] W. Zwonarz, Trajectory planning and path tracking in real time, Design, development and implementation of real-time systems - Scientific Papers of the Polish Information Processing Society Scientific Council (2013), 191–200
[3] W. Zwonarz, Time stability of computer generated control in real-time, Automatics, vol. 17 no. 2, (2013), 271–277
[4] https://pl3a.mitsubishielectric.com/fa/pl/products/rbt/robot/
[5] https://www.raspberrypi.org/
[6] https://rt.wiki.kernel.org

This work is supported by AGH University of Science and Technology, grant no. 11.11.120.396

Part V
Distributed Systems & Cloud Computing

Chapter 10

Rybu: Imperative-style Preprocessor for Verification of Distributed Systems in the Dedan Environment

1. Introduction

In the education in computer science, one of important skills concerns parallel programming and synchronization, especially in distributed environment. Yet, there are difficulties in teaching this issues in undergraduate studies, where students do not have sound experience and mathematical basis. Students do not have proper sense of importance of proper synchronization, and teachers do not have the tools supporting proper evaluation of students' work. Without such tools, it is hard to track a parallel program and show errors or inconsistences in student's solution.

There are many methods for concurrent systems verification, including proving methods, bisimulation and model checking. Static methods are used in design phase, while dynamic ones during run-time. For evaluation of students' work, static methods are mostly applicable, but sometimes a possibility to simulate a program is very help-ful. Yet, most verification formalism, like Petri net analysis or model checking tech-nique requires some knowledge from the user, not typically possessed in early stages of studies in computer science. For the same reason, specification formalism like CSP [1] or CCS [2] may be introduced rather on older years of study.

Formal verification is widely used in computer systems development, however, the authors' experience with formal methods showed that designers are willing to use the results of formal verification carried out by another person, but they are reluctant to learn a formalism (for example, model checking). Many designers, especially students of first years of study, are not prepared to use advanced techniques, for example:

"… those who taught mathematical modeling (or "formal methods") faced daunting challenges. First, most modeling tools used seemingly esoteric notations that were hurdles for many students …" [3]

Formal methods give significant help if they may be used in automatic way, not requiring specialistic knowledge from the users. Automatic verification tool Alloy was built in MIT for the purpose of verification in student laboratory [4]. In many verifiers, automatic deadlock detection concerns total deadlocks, in which all processes participate [5]. In other approaches, partial deadlocks may be automatically checked, but only in systems specific process structure [6], [7]. Alternatively, partial deadlocks may be identified using temporal formulas related to individual features of a model [8].

In Institute of Computer Science, WUT, a system Dedan (Deadlock analyzer, [9]) for modeling and automatic verification of distributed systems is used in student laboratory. Students model their solutions in IMDS (Integrated Model of Distributed Systems [10], [11]) formalism and check them against deadlocks. Although Dedan is based on temporal logic and model checking, universal formulas are used to find deadlocks and the process of temporal verification is hidden inside the tool. The universality of formulas lays in the fact that they do not depend on the structure of the model. Both total and partial deadlocks may be identified. A user simply programs a solution in Dedan input format and checks the correctness in "push the button" manner. Dedan confirms the deadlock freeness or gives a counterexample in readable form of sequence diagram-like graph. A counterexample is a sequence of states of servers and messages sent between them, so the analysis of the counterexample helps in locating a source of error.

The Dedan tool offers additional analysis features like observation of the total graph of a verified system, simulation of the system over this graph, over a counterexample or over individual components of the system, where the components are converted to automata-like graphs.

The disadvantage of Dedan verification environment is the necessity of specification of a model in Dedan input language. This language is derived from the formal definition of IMDS, using sets of *servers*, *agents* (distributed computations), servers' state *values* and servers' *services*. Behavior of a system is defined as sets of actions "glued" by servers' *states*, or alternatively by or agents' *messages*. Actions grouped in one of these two manners form the processes: a server process contains all actions of this server (with states of this server on input) wile an agent process contains all actions of this agent (with messages of this agent in input). Thus, this specification style is far from typical programming style and is exotic to students.

Two authors of this paper (Maciej Bielecki and Jan Michalski), students of ICS WUT, developed a preprocessor called Rybu [12], which overcomes this drawback, and simplifies the specification using imperative programming-like input language ([13]-Ch. 2), with its syntax similar to a subset of C language. What is more, the language of Rybu has much higher level of abstraction and is able to generate large models from simple specifications due to Cartesian products of values used in servers source code. It is impossible to build such large models manually.

The elaboration of higher-level programming language for distributed systems specification, and Rybu preprocessor for Dedan environment, are contributions of this paper.

The paper is organized as follows: Section 2 contains an overview of IMDS formalism. Section 3 presents an example of a verification under Dedan. In Section 4 main ideas of Rybu are presented. Rules of conversion of Rybu constructs to Dedan input are given in Section 5. Section 6 contains an example of advanced verification using Rybu. Conclusions and future directions of Dedan and Rybu development are covered in Section 7.

2. Integrated Model of Distributed Systems

Typically, a distributed system is modeled as a set of *servers* exchanging messages. Such a view follows a *client-server* paradigm [14]. On the other hand, processes traveling between servers are used in *remote procedure calling* paradigm (RPC [14]). Such travelling processes will be called *agents* in the paper, and a called procedure will be called a *service*, which is a typical term in RPC view. In the Integrated Model of Distributed Systems both views are included in uniform formalism.

The servers' states are the pairs (*server, value*). Also, the servers offer sets of services which may be invoked by the agents' messages. Therefore the messages are triples (*agent, server, service*).

The system behavior is built over *actions*, which have a form of relation (*message, state*) λ (*new_message, new_state*). A special kind of action terminates an agent: a new message is absent. The actions are executed in interleaving manner [15]. The processes are extracted from the specification of a system: *server processes* group the actions with common server's states on their input (and on output as well). *Agent processes* group the actions with common agent's messages on their input (and on output, if present). There are two possible decompositions of a system: a decomposition into server processes is the *server view*, while decomposition into agent processes is the *agent view* [10].

A *configuration* of the system consists of all servers' states and all agents' messages (except terminated ones). An *initial configuration* consists of *initial states* and *initial messages*. The semantics of a system is defined by a Labeled Transition System (LTS, [16]). Configurations are the nodes of LTS (initial configuration is the initial node) and actions are the transitions of LTS. The formal definition of IMDS is given in the Appendix, and it is described in [11].

The important feature of IMDS is asynchronous specification, which it exploits natural features of distributed systems:

- *locality* of action in servers: any action of a server is executed on a basis of current state of this server and messages pending at this server, no notion like global state agreed between servers may enable of prohibit any action (which is the case in synchronous formalisms like CSP [1] or CCS [2]),
- *autonomy* of servers: a server itself decides which one of the enabled actions will be executed first, no external element can influence such decisions,
- *asynchrony of actions*: a server's state waits for matching messages or a pending message waits for a matching state, no elements occur in synchronized way,
- *asynchrony of communication*: messages are passed over unidirectional channels between servers without any synchronization of sending a message along a channel with receiving it; the only way to send a reply is to send a response message over a second, independent channel in reverse direction; synchronous models like CSP or CCS assume some knowledge about global state of a system to perform inter-process communication, which is unrealistic in distributed systems.

In Lotos, asynchronous specification is used, but verification typically concerns total deadlocks [14]. Specification in IMDS allows for automatic partial or total deadlock

detection in a modeled system. It is achieved by using universal (not related to a structure of verified system) temporal formulas over IMDS elements: states and messages [11].

3. Example verification in Dedan – two semaphores

The example of deadlock detection is presented on the system of two threads *proc1* and *proc2* using two semaphores *sem1* and *sem2*. The agents come from their own servers, and each semaphore has its own server. The pseudo-code of the system is:

```
proc1:          proc2:
sem1.wait;      sem2.wait;
sem2.wait;      sem1.wait;
sem1.signal;    sem2.signal;
sem2.signal;    sem1.signal;
stop            stop
```

This system falls into a total deadlock when *proc1* holds *sem1* and waits for *sem2* and *proc2* holds *sem2* and waits for *sem1*. The system is presented below in IMDS notation, which is the input notation of Dedan. The specification is in the server view. It is simply a grouping of actions on individual servers. A system is defined as a sequence of server type specifications (enclosed by ***server*** *… }*; – lines 2-9, 10-19*)*, server and agent instances (variables) declaration (***agents*** *…*, ***servers*** … – lines 20,21) and an initial configuration phrase (***init*** → *{…}* – lines 22-27). A server type heading (l. 2,10) contains a set of formal parameters: agents and servers used in actions of the server type. Formal parameters may be vectors, as *A[2]* and *proc[2]* (l.2). Each server has a set of services (l.3,11), a set of states (l.4,12) and a set of actions assigned (in arbitrary order, l.6-8, 14-18). Services and states may be vectors. For a compact definition, repeaters may precede the actions in a server type definition (l.6-8). A repeater is an integer variable with defined range. The definition of an action is repeated for every value of a preceding repeater. If many repeaters are applied, a Cartesian product of their values is used. The indices of agents, states and services indicate individual instances (l.6-8).

Server and agent variables may be organized in vectors (l.20,21). In initialization part (l.22), actual parameters are bounded to formal parameters (again, repeaters and indices may be used – l.23-26). The initial states of servers and the initial messages of agents are also assigned.

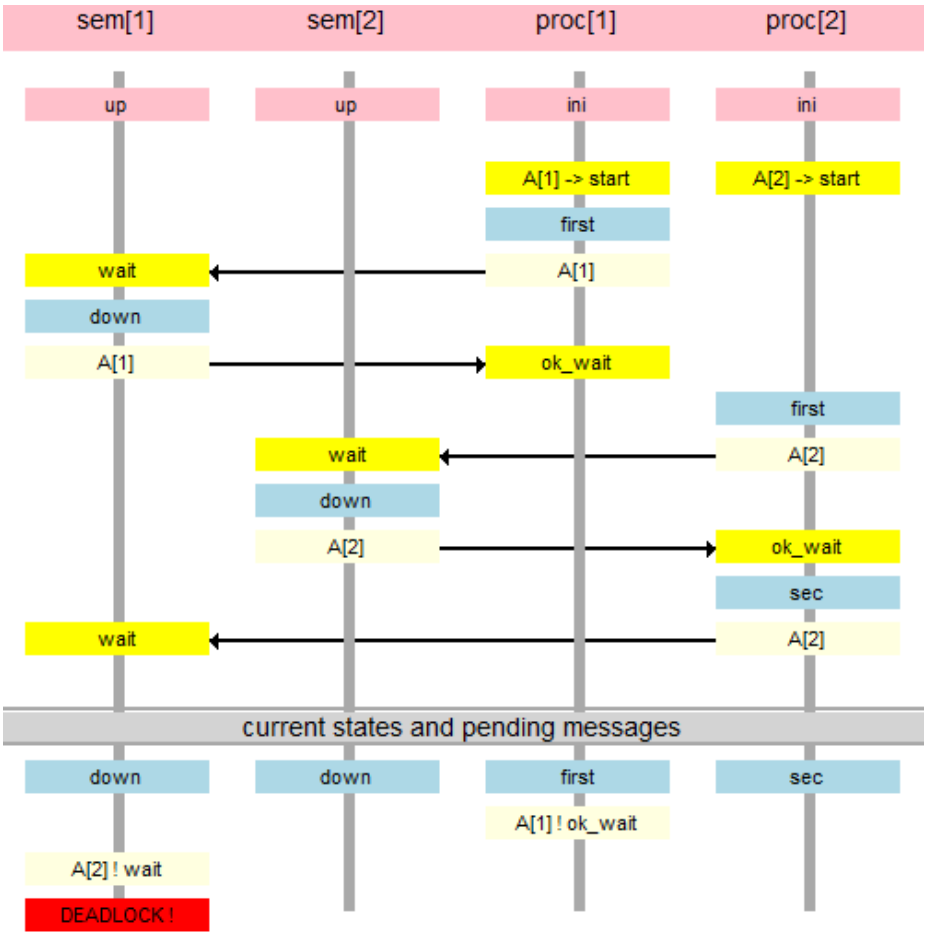

Figure 1. The counterexample for a deadlock found in the two_sem system

```
1. system two_sem;

2. server:  sem (agents A[2]; servers proc[2]),
3. services          {wait, signal},

4. states   {up, down},

5. actions {
6. <j=1..2> {A[j].sem.wait, sem.up} -> {A[j].proc[j].ok_wait, sem.down},
7. <j=1..2> {A[j].sem.signal, sem.down} -> {A[j].proc[j].ok_sig, sem.up},
8. <j=1..2> {A[j].sem.signal, sem.up} -> {A[j].proc[j].ok_sig, sem.up},
9. };

10. server: proc (agents A; servers sem[2]),
11. services          {start, ok_wait, ok_sig},
12. states   {ini, first, sec, stop},
```

```
13. actions{
14.         {A.proc.start, proc.ini} -> {A.sem[1].wait, proc.first},
15.         {A.proc.ok_wait, proc.first} -> {A.sem[2].wait, proc.sec},
16.         {A.proc.ok_wait, proc.sec} -> {A.sem[1].signal, proc.first},
17.         {A.proc.ok_sig, proc.first} -> {A.sem[2].signal, proc.sec},
18.         {A.proc.ok_sig, proc.sec} -> {proc.stop},
19. };

20. agents:         A[2];
21. servers:        sem[2], proc[2];

22. init ->   {
23. <j=1..2>        A[j].proc[j].start,
24.         proc[1](A[1],sem[1],sem[2]).ini;
25.         proc[2](A[2],sem[2],sem[1]).ini;
26. <j=1..2>        sem[j](A[1],A[2],proc[1],proc[2]).up,
27. }.
```

The verification is performed in the Dedan environment. An obvious deadlock is found, which counterexample is shown in Fig. 1.

4. Overview of the Rybu preprocessor

The main purpose of Rybu is to simplify verification in Dedan. Therefore, a limited class of systems may be modeled, but the usage should be more convenient compared to IMDS notation. Namely, modeled systems are organized within client-server paradigm [15]. Other kinds of systems, for example modeling vehicle guidance systems, where servers model road segment controllers and agents model the vehicles [16], cannot be specified in Rybu.

A basis of Rybu is the server view of a modeled system. A Rybu system consists of two kinds of servers: reactive servers, similar to the semaphores in the example, which serve as resources and offer their states and services to other servers. From now on, the term *server* will be used in the meaning reactive servers. If we mention a server in IMDS formalism, it is named *IMDS server*.

The second kind of servers performs distributed computations, called *threads*. The agents origin and run on threads, calling the services offered by the servers.

4.1. Servers

The services of servers are invoked by threads by means of messages. A server consists of *state variables* and *actions*. State variables are implemented as sets of IMDS servers' *values*, and actions are implemented as sets of IMDS actions. Actions define how the invocations of the servers' services, which may be regarded as procedure calls from the threads, are executed. When such a service is called, the server changes the values of its state variables and returns a response (being a message itself). *Return values* – technically threads' resources – are atoms of an enumeration type bound to specific action, which is defined by usage.

Rybu action is a 4-tuple of (*service*, *predicate*, *return value*, *state update*) where:

- *service* – an identifier used by a thread to trigger a server's action,
- *predicate* (optional) – a condition (over state variables) on which the action may be executed,
- *return value* – a value returned to the calling thread,
- *state update* (optional) – a function mapping a server's state to a new state of the server, represented in the syntax as assignment to one or more state variables.

The server type `sem` from the `two_sem` example has the following form in Rybu:

```
1. server sem {
2.      var state : {up, down};
3.      { wait | state == :up } -> { state = :down; return :ok; }
4.      { signal } -> { state = :up; return :ok; }
5. }
```

The service `wait` may be executed on condition that the state of the semaphore is `:up`, then it is updated to `:down` and the response `:ok` is issued to the caller. The service `signal` may be executed any time, it leaves the state of the semaphore `:up` and confirms it by the `:ok` response.

4.2. Actions

Each action (*service*, *predicate*, *return value*, *state update*) is applied to a *service*, for the value of *state variables* for which the *predicate* evaluates to true. If the predicate is true, the action causes state variables to receive the values appointed by *state update* function and a response if issued with *return value* to a calling *thread*.

Note that for a specific *service*, more than one *action* may be defined, with different *state updates*, and intersecting set of *state variables* values. This causes nondeterministic state transitions of *state variables*. In this case the actions are nondeterministic.

4.3. Threads

The actions in IMDS server are not set up in any sequence, it is only a set of actions and it may be listed in arbitrary order. It is a case because there is no notion comparable to a "program counter", which causes the instructions to be executed in a sequence. In Rybu, the instructions are threaded in the order of placement as in typical imperative programming language ([13]-Ch.2). Additional structural instructions may change the order of execution, for example *loop* statement or choice (*match*) statement. The IMDS server `proc[1]` with its agent `A[1]` are modeled in Rybu as:

```
1. var sem1 = sem() { state = :up };
2. var sem2 = sem() { state = :up };
3. thread proc1() {
4.      sem1.p();
5.      sem2.p();
6.      sem1.v();
7.      sem2.v();
8. }
```

Variables *sem1* and *sem2* are instances of the sever type *sem*. The thread *proc1* calls the services *p* and *v* of the server type *sem* in the order just taken from the pseudo-code at the beginning of Section 3. The IMDS server *proc*[2] with its agent *A*[2] is very similar, except that the order of operations on the semaphores is switched (*sem2*, then *sem1*).

4.4. Statements

The Rybu preprocessor uses single executive statement ***match***. It consists in sending a message to a server (calling its service) and waiting for response (reply message). Then, a response value may direct the control to a proper follow-up statement. The *server.service();* statement form, used in Section 4.3, is a simplified version. It is used when the service returns only :*ok* response value. The example of the full version of ***match*** statement is contained in Section 5.3.

There is also the control statement ***loop*** *{... statements ...}*.

4.5. Communication in Rybu

Communication in Rybu follows the scheme of a remote procedure call [15]: a service of a server is invoked by a thread, then it waits for a response inactively. The response may contain a value of an enumeration type.

4.6. Data types

The set of possible server states is the Cartesian product of ranges of all of its *state variables*. Data types are necessary to determine the range of possible state variable values. Each type denotes a finite set of values. There are three types available:

1. Integer range: *min..max* (where *min* and *max* are integer-valued expressions) -- values of this type are just integers in range [*min; max*].
2. Enumeration: {*atom1, atom2*} - values of an enumeration are single atoms, e.g., :*atom1*. Note that the atoms are preceded by a colon when used.
3. Vectors of the above types: *elem_type*[*length*] - values are concrete vectors of values of *elem_type*, holding the *length* size, e.g., for the vector type (0..3)[4], a possible vector value could be [0,3,1,2] .

4.7. Organization of Rybu

The program consists of three main parts:

1. *Parsing*. A recursive descent parser is used, implemented using Monadic Parser Combinators [17]. This process generates an Abstract Syntax Tree [18].
2. *Conversion*. Rybu servers and threads are converted to state machines, represented by sets of state transitions.
3. *Output generation*. The resulting automata are printed as IMDS servers in Dedan syntax.

The implementation in Haskell is covered by a suite of unit tests and a set of (input file, expected output) pairs. The latter can be expensive to maintain, as the test result is sensitive to the ordering of actions produced by the program (which has no semantic meaning).

5. Conversion rules

In this section, implementation all constructs of Rybu in IMDS is presented: servers, actions and variables in servers, threads and messages.

5.1. Servers and servers' variables

Each Rybu *server* is converted to an IMDS *server type*. The set of its IMDS *states* is defined by a Cartesian product of sets of all possible values of the *state variables* declared in the server.

For example, Rybu server *test*:

```
1. server test {
2.      var val1: 1..3;
3.      var val2: {true, false};
4. }
```

generates an IMDS server that has the following set of *states*:

```
{val1_1_val2_true, val1_1_val2_false, val1_2_val2_true,
val1_2_val2_false, val1_3_val2_true, val1_3_val2_false}
```

Proper server instances are declared, and in the initialization part, state variables are initialized, for example for the declaration:

```
var t = test() {val1 = 1; val2 = :false};
```

the IMDS declaration is:

```
servers t:test,…
```

and the IMDS initialization is:

```
t(… used servers and agents …).val1_1_val2_false,
```

5.2. Actions in server

An action (*service*, *predicate*, *return value*, *state update*) is converted to a set IMDS actions applied to a *service*, for all possible values of *state variables* for which the

predicate is true. For every pair of *(service, state variables values)*, for which the predicate is evaluated to true, an IMDS action is generated with the following properties:

- The IMDS *input state* is the one of which the predicate was evaluated to true.
- The IMDS *input message* calls the *service*.
- An *output state* is obtained by application of the *state update* function to the input state (state variables mentioned get their new values while all other state variables preserve their values).
- The above two steps are applied to each IMDS *agent* implementing a *thread* that invokes the service (see Section 5.3 Treads).
- A *return value* is sent and IMDS *output message* to the IMDS server implanting the *thread* which calls the action.

For example, the action 4 in the example in Section 4.1 is converted to a following set of IMDS actions, provided that two threads `t1` and `t2` invoke the service `signal`. The example thread `t1` is implemented as IMDS server `S_t1` and its agent `A_t1`, similarly `t2`.

```
{A_t1.sem.signal, sem.up} -> {A_t1.S_t1.ok, sem.up},
{A_t1.sem.signal, sem.down} -> {A_t1.S_t1.ok, sem.up},
{A_t2.sem.signal, sem.up} -> {A_t2.S_t2.ok, sem.up},
{A_t2.sem.signal, sem.down} -> {A_t2.S_t2.ok, sem.up},
```

5.3. Threads

Each *thread* is converted into an IMDS server and an accompanying agent. A *thread* issues messages invoking some servers' services and then it waits for a response corresponding to this call. States of a thread are defined by the "position in code" (a virtual "program counter"). From each such state there are as many IMDS actions generated as the number of branches in the `match` statement, each reacting to a specified response. Below an example of Rybu code shows thread syntax with ***match*** statement which branches depending on response received from calling the service `y` on the server `s1`.

```
1. thread x() {
2.       loop {
3.                match s1.y(){
4.                   :ok => s2.z();
5.                   :er => s3.v();
6.             }
7.       }
8. }
```

In IMDS, every statement is associated with its own value of an IMDS server's state, which acts as a „program counter". The above example is converted to a set of IMDS actions:

```
{A_x.S_x.ok, S_x.s0_s1_y} -> {A_x.s2.z, S_x.s2_s2_z},
{A_x.S_x.er, S_x.s0_s1_y} -> {A_x.s3.v, S_x.s5_s3_v},
{A_x.S_x.ok, S_x.s2_s2_z} -> {A_x.s1.y, S_x.s0_s1_y},
{A_x.S_x.ok, S_x.s5_s3_v} -> {A_x.s1.y, S_x.s0_s1_y},
```

States *s0_s1_y, s2_s2_z, ...* are used as a „program counter". Every IMDS server implementing a thread starts from *s0_...* state. In IMDS initialization part, the IMDS server is initialized and its agent executes its first message in the thread's code:

```
S_x(s1,s2,s3,A_x).s0_s1_y,
   A_x.s1.y,
```

5.4. Communication

The communication in Rybu follows the remote procedure call principle [15]. A call consists of two messages: one to call a server and the other one that carries a response. A thread calls a server by issuing a message that invokes the server's service (for example *A_x.s2.z* above). A response is a message in the opposite direction, sent from the server to the calling thread, like *A_x.S_x.er*. The pair of messages is passed in the context of the thread's agent *A_x*.

6. Advanced verification in Rybu

Consider a system consisting of two bounded buffers. The buffers are modeled by the counters representing current numbers of elements – the contents are irrelevant for synchronization purposes.

There are two threads continuously trying to balance the buffers – they move items from a buffer that is more filled to the other one. The threads are symmetric, but every thread checks different direction of moving elements.

A deadlock happens because a check and an action are not guarded by a critical section, for example when *shouldPut1* check causes the *put2* action. This allows for interleaving that may cause *shouldPut* checks to happen in both user threads first, and then both threads *put* elements to the same buffer, but there is a room only for one element. In this case both threads block on the same semaphore waiting for an element to appear in the other buffer, but there is no third thread running to put an element there: a deadlock occurs. The following code shows an implementation of this model in Rybu.

```
const N=3;

server SemN {
    var value: 0..N;

    { signal | value < N } -> { value = value + 1; return :ok; }
    { signal | value == N } -> { return :ok; }

    { wait | value > 0 } -> { value = value - 1; return :ok; } }
}

server Buf {
    var count1: 0..N;
    var count2: 0..N;
```

```
    { shouldPut1 | count1 - count2 <= 0 } -> { return :true; }
    { shouldPut1 | count1 - count2 > 0 }  -> { return :false; }

    { shouldPut2 | count2 - count1 <= 0 } -> { return :true; }
    { shouldPut2 | count2 - count1 > 0 }  -> { return :false; }

    { put1 | count1 < N } -> { count1 = count1 + 1; return :ok;}
    { get1 | count1 > 0 } -> { count1 = count1 - 1; return :ok;}
    { put2 | count2 < N } -> { count2 = count2 + 1; return :ok;}
    { get2 | count2 > 0 } -> { count2 = count2 - 1; return :ok;}
}

var buf = Buf() { count1 = 0, count2 = 0 };
var sBuf1 = SemN() { value = 0 };
var sBuf2 = SemN() { value = 0 };

thread User1() {
    loop {
       match buf.shouldPut1() {
          :true  => {
             buf.put1();
             sBuf1.signal();
             sBuf2.wait();
             buf.get2();
          }
          :false => {
             buf.put2();
             sBuf2.signal();
             sBuf1.wait();
             buf.get1();
          }
       }
    }
}

thread User2() {
    loop {
       match buf.shouldPut2() {
          :true  => {
             buf.put2();
             sBuf2.signal();
             sBuf1.wait();
             buf.get1();
          }
```

```
            :false => {
               buf.put1();
               sBuf1.signal();
               sBuf2.wait();
               buf.get2();
            }
         }
      }
}
```

The example illustrates the use of multiple state variables (two range variables in the `Buf` server) and imperative programming of the threads.

The deadlock may be fixed by introducing a mutual exclusion semaphore that allows only one thread at a time to perform `shouldPut` check and `put` operation in a critical section.

7. Conclusions and future work

The verification of synchronization projects, performed by the students, prove the usefulness of verification in the Rybu/Dedan environment. Students appreciate automaticity of verification, which does not require from them any knowledge on temporal logics and model checking.

Rybu allows to prepare larger models, and on higher level of abstraction. Many cases were checked, including large ones, impossible to model directly in Dedan. For example, 131 lines of Rybu code of a student's solution generates about 6,000 of Dedan code. Till now, no student project exceeding available memory occurred. However for such a possible case, export of IMDS specification to external verifiers using reachability space reduction was prepared (for example Uppaal [19]).

Some other examples were verified, they may be found in [20], [16]. Also, a verification of a large model of the Karlsruhe Production Cell benchmark [21] was performed [22]. Unlike in other approaches, simple asynchronous protocols are used between distributed controllers of individual road segment controllers [16] or Production Cell devices [22]. Such a modeling is suitable for Internet of Things approach (IoT [23]), where independent devices autonomously negotiate their common behavior using simple protocols. Rybu/Dedan couple allows for rapid prototyping approach of distributed solutions, because of automatic verification of prepared models.

Also, real-time constraints may be applied to action and communication time delay for verification of real-time systems [24]. Dedan is equipped with real-time constraints which may be imposed on actions and communication. Such constraints may be imported to Rybu.

Rybu is based on client-server paradigm, but other approaches are possible. For example, agents in the system may travel through the servers, not returning to the servers they origin from. This is a subject for further research concerning a different kind of input language, addressed to traveling agents.

In Rybu some improvements are needed, for example state variables in threads, thread types, enriched communication statements.

References

[1] C.A.R. Hoare, Communicating sequential processes. Commun. ACM. 21(8), 666–677 (1978). doi: 10.1145/359576.359585

[2] R. Milner, A Calculus of Communicating Systems. Springer-Verlag, Berlin Heidelberg (1984). ISBN: 0387102353

[3] M.J. Lutz, Modeling software the alloy way. In: 2013 IEEE Frontiers in Education Conference (FIE), Oklahoma City, OK, 23-26 October 2013. p. 3. IEEE (2013). doi: 10.1109/FIE.2013.6684771

[4] M.C. Reynolds, Lightweight Modeling of Java Virtual Machine Security Constraints. In: 2nd International Conference on Abstract State Machines, Alloy, B and Z, ABZ 2010, Orford, QC, Canada, 22-25 February 2010. pp. 146–159. Springer, Berlin, Heidelberg (2010). doi: 10.1007/978-3-642-11811-1_12

[5] N. Kokash, F. Arbab, Formal Behavioral Modeling and Compliance Analysis for Service-Oriented Systems. In: de Boer, F.S. (ed.) Formal Methods for Components and Objects - 7th International Symposium, FMCO 2008, Sophia Antipolis, France, 21-23 October, 2008, LNCS 5751,. pp. 21–41. Springer, Berlin Heidelberg (2008). doi: 10.1007/978-3-642-04167-9_2

[6] Y. Yang, X. Chen, G. Gopalakrishnan, Inspect: A Runtime Model Checker for Multithreaded C Programs, Report UUCS-08-004. University of Utah, Salt Lake City, UT (2008). http://www.cs.utah.edu/docs/techreports/2008/pdf/UUCS-08-004.pdf

[7] D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, K. Wolf, Analysis on demand: Instantaneous soundness checking of industrial business process models. Data Knowl. Eng. 70(5), 448–466 (2011). doi: 10.1016/j.datak.2011.01.004

[8] J. Arcile, T. Czachórski, R. Devillers, J.-Y. Didier, H. Klaudel, A. Rataj, Modelling and Analysing Mixed Reality Applications. In: Gruca, A., Brachman, A., Kozielski, S., and Czachórski, T. (eds.) Man-Machine Interactions 4: 4th International Conference on Man-Machine Interactions, ICMMI 2015 Kocierz Pass, Poland, 6-9 October 2015. pp. 3–17. Springer International Publishing, Cham, Switzerland (2016). doi: 10.1007/978-3-319-23437-3_1

[9] Dedan, http://staff.ii.pw.edu.pl/dedan/files/DedAn.zip

[10] S. Chrobot, W.B. Daszczuk, Communication Dualism in Distributed Systems with Petri Net Interpretation. Theor. Appl. Informatics. 18(4), 261–278 (2006). https://taai.iitis.pl/taai/article/view/250/taai-vol.18-no.4-pp.261

[11] W.B. Daszczuk, Communication and Resource Deadlock Analysis using IMDS Formalism and Model Checking. Comput. J. 60(5), 729–750 (2017). doi: 10.1093/comjnl/bxw099

[12] Rybu, https://zyla.neutrino.re/rybu/.

[13] R.W. Sebesta, Concepts of Programming Languages. Addison-Wesley Publishing Compan, Reading, MA (1996). ISBN: 10: 0-13-139531-9

[14] W. Jia, W. Zhou, Distributed Network Systems. From Concepts to Implementations. Springer, New York (2005). ISBN: 0-387-23839-5

[15] P. Godefroid, P. Wolper, Using partial orders for the efficient verification of deadlock freedom and safety properties. In: 3rd International Workshop, CAV '91, Aalborg, Denmark, 1-4 July, 1991, LNCS 575. pp. 332–342. Springer, Berlin Heidelberg (1992). doi: 10.1007/3-540-55179-4_32

[16] M.A. Reniers, T.A.C. Willemse, Folk Theorems on the Correspondence between State-Based and Event-Based Systems. In: 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. pp. 494–505. Springer-Verlag, Berlin Heidelberg (2011). doi: 10.1007/978-3-642-18381-2_41

[17] S. Djaaboub, E. Kerkouche, A. Chaoui, A New Approach for Generating LOTOS Specifications from UML Dynamic Models. In: Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering - C3S2E '15, 13 - 15 July 2015, Yokohama, Japan. pp. 138–143. ACM Press, New York, NY (2008). doi: 10.1145/2790798.2790813

[18] B. Czejdo, S. Bhattacharya, M. Baszun, W.B. Daszczuk, Improving Resilience of Autonomous Moving Platforms by real-time analysis of their Cooperation. Autobusy-TEST. 17(6), 1294–1301 (2016).

[19] G. Hutton, E. Meijer, Monadic parsing in Haskell. J. Funct. Program. 8(4), 437–444 (1998). doi: 10.1017/S0956796898003050

[20] G. Fischer, J. Lusiardi, J. Wolff von Gudenberg, Abstract Syntax Trees - and their Role in Model Driven Software Development. In: International Conference on Software Engineering Advances (ICSEA 2007), Cap Esterel, France, 25-31 August 2007. pp. 38–38. IEEE (2007). doi: 10.1109/ICSEA.2007.12

[21] G. Behrmann, A. David, K.G. Larsen, A Tutorial on Uppaal 4.0. Report, University of Aalborg, Denmark (2006). http://www.uppaal.com/admin/anvandarfiler/filer/uppaal-tutorial.pdf

[22] Dedan Examples, http://staff.ii.pw.edu.pl/dedan/files/examples.zip
[23] C. Lewerentz, T. Lindner (eds.), Formal Development of Reactive Systems. Case Study Production Cell. LNCS 891, Springer-Verlag, Berlin, Heidelberg (1995). doi: 10.1007/3-540-58867-1
[24] W.B. Daszczuk, Asynchronous Specification of Production Cell Benchmark in Integrated Model of Distributed Systems. In: 23rd International Symposium on Methodologies for Intelligent Systems, ISMIS 2017, Warsaw, Poland, 26-29 June 2017. Springer-Verlag, Berlin Heidelberg (2017).
[25] G.M. Lee, N. Crespi, J.K. Choi, M. Boussard, Internet of Things. In: Evolution of Telecommunication Services, LNCS 7768. pp. 257–282. Springer-Verlag, Berlin Heidelberg (2013). doi: 10.1007/978-3-642-41569-2_13
[26] H. Kopetz, Internet of Things. In: Real-Time Systems. Design Principles for Distributed Embedded Applications. pp. 307–323. Springer US (2011). doi: 10.1007/978-1-4419-8237-7_13

Appendix – IMDS formal definition

$S =$ $\{s_1,s_2,...,s_{card(S)}\}$ – a finite set of servers
$A =$ $\{a_1,a_2,...,a_{card(A)}\}$ – a finite set of agents
$V =$ $\{v_1,v_2,...,v_{card(V)}\}$ – a finite set of values
$R =$ $\{r_1,r_2,...,r_{card(R)}\}$ – a finite set of services
$P \subset$ $S \times V$ – a set of states
$M \subset$ $A \times S \times R$ – a set of messages
$H =$ $P \cup M$ – a set of items
$T \subset$ H;
$\forall_{p,p' \in T \cap P}\ p = (s,v), p' = (s',v'),\ p \neq p' \Rightarrow s \neq s'$;
$\forall_{m,m' \in T \cap M}\ m = (a,s,r),\ m' = (a',s',r'),\ m \neq m' \Rightarrow a \neq a'$ – a con guration; one state for every server, at most one message for every agent
$T_0 \subset$ H;
$\forall_{s \in S} \exists_{p \in T_0 \cap P}\ p = (s,v), v \in V$;
$\forall_{a \in A} \exists_{m \in T_0 \cap M}\ m = (a,s,r),\ s \in S,\ r \in R$ – initial con guration; one state for every server, one message for every agent
$\Lambda =$ $\{(m,p)\lambda(m',p')\} \cup \{(m,p)\lambda(p')\} \mid$
$m = (a,s,r) \in M,\ m' = (a',s'',r') \in M,\ a' = a$;
$p = (s,v) \in P,\ p' = (s',v') \in P,\ s' = s$ – a set of actions – ordinary ones and agent-terminating ones
$T_{inp}(\lambda) =$ $T \mid T \supset \{m,p \mid (m,p)\lambda(m',p')$ or $(m,p)\lambda(p')\}$ – an input con guration of the action λ; contains the input items of the action
$T_{out}(\lambda) =$ $T \mid T \supset \{m',p' \mid (m,p)\lambda(m',p')\}$ *or*
$T \supset \{p' \mid (m,p)\lambda(p')\} \wedge (m = (a,s,r), \forall_{m'=(a',s',r') \in M}\ a' = a \Rightarrow m' \notin T)$ – an output con guration of the action λ; contains he output items of the action, does not contain an output message of the agent in a case of an agent-terminating action
$\Delta =$ $\{\delta(\lambda) \mid \lambda \in \Lambda,\ T_{inp}(\lambda)\delta(\lambda)T_{out}(\lambda),$
$(\ ((a,s,r),(s,v))\lambda((a,s',r'),(s,v')) \vee ((a,s,r),(s,v))\lambda((s,v'))\ ) \wedge$
$\forall_{a'' \in A}\ a'' \neq a \wedge (a'',s'',r'') \in T_{inp}(\lambda) \Rightarrow (a'',s'',r'') \in T_{out}(\lambda) \wedge$
$\forall_{s''' \in S}\ s''' \neq s \wedge (s''',v''') \in T_{inp}(\lambda) \Rightarrow (s''',v''') \in T_{out}(\lambda)\ \}$ – a succession relation; replaces the input items of an action by the output items, all other items are preserved
$LTS =$ $< N,N_0,W > \mid$
$N = \{T_0,T_1,...\}$ (nodes);
$N_0 = T_0$ (initial node);
$W = \{\delta(\lambda) \in \Delta \mid \lambda \in \Lambda,\ T_{inp}(\lambda)\ \delta(\lambda)\ T_{out}(\lambda) \wedge$
$(T_{inp}(\lambda) = T_0 \vee \exists_{\lambda' \in \Lambda}\ \delta(\lambda') \in W,\ T_{out}(\lambda') = T_{inp}(\lambda))$ (transitions) $\}$ – the labeled transition system

$B(s) =$ $\{ \lambda \mid ((a,s,r),(s,v))\lambda((a,s',r'),(s,v')),\ \lambda \in \Lambda,\ s' \in S,\ a \in A,\ v,v' \in V,\ r,r' \in R \}$ – server process of a server $s \in S$

$C(a) =$ $\{ \lambda \mid ((a,s,r),(s,v))\lambda((a,s',r'),(s,v')),\ \lambda \in \Lambda,\ s,s' \in S,\ v,v' \in V,\ r,r' \in R \}$ – agent process of an agent $a \in A$

$\boldsymbol{B} =$ $\{B(s)\} \mid \forall_{s \in S}\, B(s) \in \boldsymbol{B}$ – the server view; a decomposition of a system into server processes

$\boldsymbol{C} =$ $\{C(a)\} \mid \forall_{a \in A}\, C(a) \in \boldsymbol{C}$ – the agent view; a decomposition of a system into agent processes

.

Chapter 11

Methods and Means of Creating Applications to Control a Complex Network Environment

1. Introduction

Contemporary IT systems are environments which are a resultant of hardware and software. Their core is a communication structure and in particular equipment and transmission media. So far, a strong division into the information processing environment and the communication environment is visible, especially from the control point of view. The communication system usually has its own autonomous system of infrastructure management, which is also dedicated to its control [1,2]. Moreover, the topology of such a system is characterized by the use of devices of many vendors ,therefore, it is very heterogeneous. The heterogeneity of the computer network is visible in several areas:

- **Protocolar.** Connection networks to perform the same tasks use a lot of different protocols. Considering, for example a routing, one can select (depending on conditions and a producer) - several protocols: IS-IS, OSFP, RIP, BGP, EIGRP etc. [3]. Most of the used protocols are open and allow for full protocol interoperability across devices of different producers. However, there are not rare situations in which a given protocol is closed and is used solely by one vendor, or implementations of a particular protocol are not fully compatible, for example, with the IEEE definition.
- **Hardware.** Network devices differ essentially in hardware capabilities and have different possibilities for implementation set of used protocols and transmission technologies. Also there are different probes for data acquisition used by devices and their resolution and a level of the access to collected data.
- **Management systems.** As a rule, every manufacturer has their own operating system that manages a particular network device. That systems differ essentially in the sphere of: system architecture, principles of their management and capacity of functional control as well. In the area of management automation there is a number of systems for collective control of devices called Network Managmet Systems (NMS). Each of these systems (e.g. HP OpenView, Alcatel-Lucent OmniVista, Cisco NMS, etc.) enables a management of heterogeneous environment of the devices, but only in a limited range. As a rule, event information is read from devices using mechanisms such as SNMP, TRAP, SFLOW NetFLOW etc. Active change of the configuration is possible for

particular NMS only if devices and NMS come from one manufacturer. Of course, it is possible to add incidentally software extensions to NMS for other manufacturers, but this involves high costs and the need to access to NMS code.

Strong heterogeneity of the environment is in contradiction to the convergence of services in the area of communication (voice, data and video traffic) as well as in application area [4,5,6]. At present, there is a strong need to create architecture of central arbiter controlling the heterogeneous environment of network devices based on standards existing on the market. The system architecture must provide the ability to easily integrate the control mechanism with applications and standards of information exchange. Such an approach to central control of the communication system is known in the literature and was successfully used in connection networks dedicated to supercomputers e.g. in multi-bus networks [7-9]. The central control application was called an arbiter, which was responsible for central information collection and controlling the whole system. The concept of arbiter, however, was appropriate for homogeneous systems. Arbiter could not be controlled by external applications. The difference between the known and the proposed arbiter structure is shown in Figure 1.

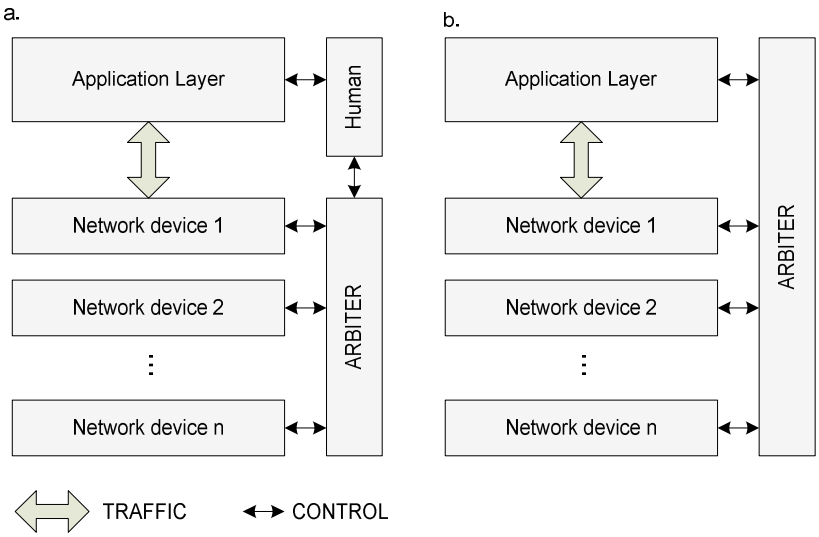

Figure 1. a. Classical arbiter architecture; b. Proposed arbiter architecture

In the classic approach (Figure 1.a.), an application layer (understood wider than it implies directly from the ISO/OSI model) communicates with a series of homogeneous network devices controlled by a dedicated protocol from a level of an application called an arbiter. The evaluation of connection network is currently determined by a human/an analyst, who modifies the parameters of the devices based on the analysis of the work of the application and on the opinions of the users. This is a typical example of reactive management of devices infrastructure that is characterized by big delay and is not appropriate for modern IT systems. In the second model (Figure 1. b.), an appropriately prepared application module is communicated to the arbiter using a dedicated channel (ssh, RestFul, etc.) [10] and in real time, depending on needs, it modifies the behavior of

the network infrastructure (QoS, Ruting, etc.) or detects anomalies and blocks suspicious traffic[11]. The arbiter communicates with devices using specialized protocols (ssh, SDN, Sflow, NetFlow etc.) [12]. Such an approach fits into the model of Software Defining Networking. In this approach, the connection network becomes a virtual component of the application. Figure 2 illustrates the model of an active control system of the connection network.

The purpose of the paper is to determine the methods and means used to create a software model for active management of a connection network, with particular regard to the methods of implementing the software components and protocols used in this model, marked with letters A and B (Figure 2.). The authors have proposed and implemented three main models of software architecture described in subsequent chapters together with criteria of their choice. The authors focused mainly on the application of the proposed architecture in the medium and small business environment (SMB). The second chapter describes the methodologies of the conducted study and the area in which the results can be used. Chapter 3 describes three different software architectures based on which the arbiter was built. Each subsection is completed with a summary and conclusions of the tests. In subsection 3.4 there is an aggregate evaluation of all three proposed architectures together with their evaluation and selection criteria.

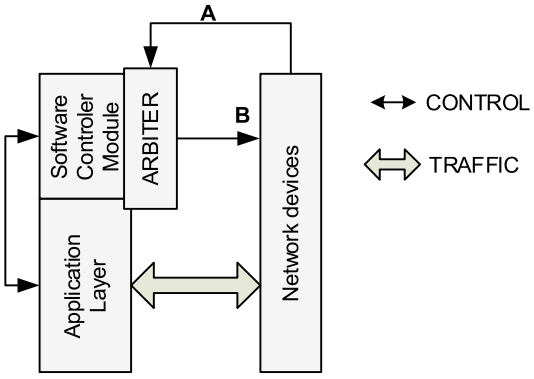

Figure 2. Model of active control of the connection network

2. Methodology of the Research

One of the main research issues during the work was to determine the ability to localise the arbiter code. The following locations were considered: network device, dedicated hardware module, software module of existing NMS system, dedicated software in SDN architecture, custom dedicated software module. Each of the listed scenarios was implemented in a laboratory environment using a real network equipment and the chosen scenarios were tested in a production environment.

The next important aspect was to determine the cost of implementation, the time needed for implementation, the level of knowledge of deployment engineers, software developers, and system administrators. At this stage, the security aspect of the solution was completely ignored, as public cryptographic techniques were used during the work.

Further work will also concentrate in this area. It should be noted that the software was implemented by groups of engineers at similar level of advancement - and every engineer could participate in the work over one scenario. The whole work was supervised in substance by two persons. They took part in every scenario.

Preliminary work has shown that there are solutions on the market that perform similar functionality, but they are internally used by companies usually in large core networks (WAN or corporate operators). Despite the attempts, the authors failed to obtain detailed information about the architecture of this software. Due to the closed architecture of these solutions their implementation potential in the SMB environment is minimal. The authors have assumed that the considered architectures must: be based on commonly used standards and technologies, thus be universal and have the potential to be implemented in SMB (limited financial outlay), have modular architecture, allow for evolutionary expansion of the system, base on relatively simple and popular software languages (this makes it easier to get engineers to administer and develop the system).

The requirements were based on interviews with 12 SMB companies. In order to determine the guidelines and requirements for this class of software, the authors took part in two internships in the companies (under the programs: „Nauka idzie w praktykę" RARR[13], "Trensferencia" RARR[14]). Companies have been operating in a IT area (inter alia: serviceprovider, training, distance learning, R&D, industry). These grants were aimed at researchers who conducted research directly in companies. One of the objectives of the program was to identify research directions important to industry as well as medium and small businesses.

3. Software Architecture to Create Applications to Control a Complex Network Environment

This chapter shows the individual tested implementation architectures of the arbiter. Finally, a summary and evaluation of each architecture are presented.

3.1. Software Defined Networking

The SDN network architecture has been described in detail in [15,16]. Its illustrative diagram is presented in Figure 3a. The SDN architecture, unlike the classic network model, moves the Control Plane layer from network devices to an external controller, where logical topologies and network flows are formed. In the next step, the flows are injected into the devices. The whole network is controlled from one place – by the controller. The controller communicates with devices using OpenFlow protocol and collects data from devices using e.g. Sflow (or other). Therefore, such architecture seems to be ideal for implementing software to central active management of the communication environment (Arbiter). Figure 3a. presents a diagram of the actual implementation of such environment in a real-world lab environment using 3 virtual machines (API-Server, Probe, Flood Light) connected together by a virtual network and connected to LAN environment (the LAN based on the Alcatel-Lucent Enterprise OmniSwitch devices of 6860 and 6900 series). The Probe server monitored network status. Applications running on the API - Server (Python, Perl, Ruby, C ++, Java, etc.) basis on the information from Probe server

sent commands to the Floodlight controller, which modified then the appropriate flows in the switches. Several tests were performed, including DDoS attack detection, QoS parameters modification for traffic coming from the application installed on the API-Server, routing change for traffic coming from the application installed on the API-Server.

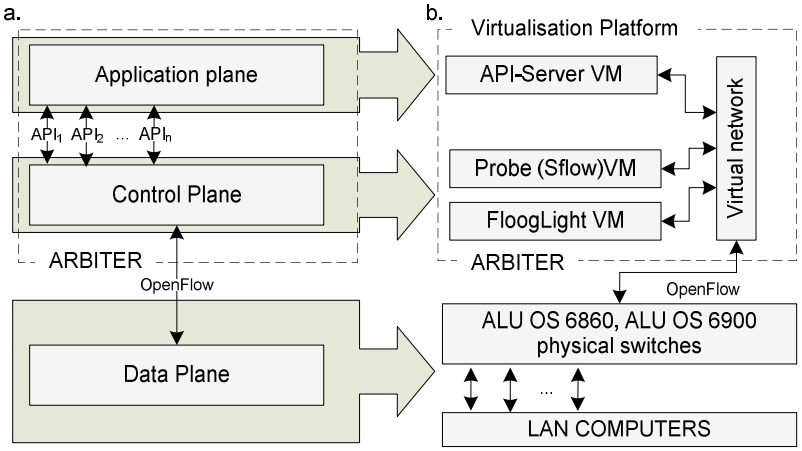

Figure 3. SDN architecture: a. model; b. implementation in laboratory environment

After the tests, the following conclusions were drawn:

- The architecture is very flexible and it enables implementation of connection environment management using the arbiter, it easily integrates with the production applications environment, it is possible to implement several programmable controllers operating in HA (High-availability cluster) mode, it uses open protocols, OpenFlow enables communication with the switch of any producer supporting the standard.
- A small portion of the enterprise class switches support fully the SDN architecture. These are switches of higher (expensive) series, located rather in the aggregation or in the core of the network layer. Analysis of the roadmap of evolution of several leading manufacturers products (Cisco, ALE, HP, Extrem Networks) has shown that most of the new switches will support in future the OpenFlow standard.
- Constructing the software controller itself is very complicated and requires from the engineers knowledge in the area of: system virtualization, network programming, computer networks, systems security.
- An improperly designed application can completely paralyze the operation of the network system, which in consequence can make the whole IT system unable to operate.
- It is necessary to introduce a very restrictive control of application modules responsible for network control.

3.2. Central Scripts

In the next architecture we tested possibility of direct control of devices using scripts written directly in the application or on a dedicated server. The general structure of the architecture is shown in Figure 4.

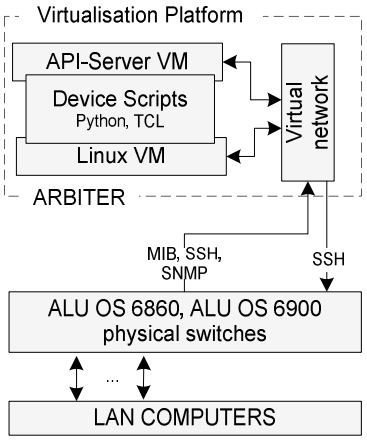

Figure 4. Arbiter architecture built on a central controller with scripts

The basic problem that had to be solved in this architecture is to develop a reliable mechanism of two-way communication between the controller and network devices. Mechanism MIB, SNMP and SSH sessions have been used for data collecting. The first two give access to all parameters of the devices and are implemented on all networked devices being produced. They are a very convenient and universal mechanism of information collecting. The communication connected with the configuration modification was carried out using SSH and commands executed remotely on the device. For this scenario, a series of tests have been performed, including modifying the current behavior of OSPF protocol (changing metrics depending on load and QoS for the given application). After the tests, the following conclusions were drawn:

- The software architecture is simple and relatively flexible. The arbiter software module can be installed on both the dedicated machine or in the applications directly. As a result of testing in lab environment, bigger stability has been demonstrated by the solution with a dedicated machine (Linux VM).
- The implementation of the architecture is relatively quick and easy, but it requires the development of dedicated scripts for devices of individual manufacturers because they differ in language syntax.
- Knowledge of the network environment, its topology is required, so the programmer must possess the knowledge of computer networks . It is possible to use an indirect solution in this area. The controller can be built on a dedicated machine (Linux VM) that will create an interface that is user friendly for application programmers and will be responsible for translating the commands sent to the devices.

- The solution, for its simplicity, allows for quick and easy failure and errors detection.
- As in the case of the SDN architecture, it is necessary to introduce a very restrictive control of modules of applications responsible for network control.

3.3. Distributed Scripts

The next architecture in which the arbiter was implemented was built on prepared scripts located on the physical network devices of different producers. The greatest inconvenience in case of implementation the architecture with central scripts was the heterogeneous interface for sending commands to network devices. The architect, application developer, must know in this case differences in CLI syntax of commands executed on the switch and their specificity. Scripting languages available on network devices (Python, BASH, TCL) were used in the next architecture (Figure 5) to create scripts called with parameters to perform specific functions or tasks. Application programmer or programmer of the dedicated controller (Linux VM) calls only the scripts on devices with specified parameters. It does not need to know the specifics of a given console syntax, because access to it has been unified by the use of scripts. Data was collected from devices in the same way as for architecture with the centrally located scripts. The environment was implemented in a lab environment and partly in a production environment. A series of tests was performed, including modifying the current behavior of OSPF and QOS protocols. After the tests, the following conclusions were drawn:

- The architecture is relatively simple to implement and does not entail the high costs associated with it. It is characterized by high stability of operation.
- The implementation of the architecture is relatively quick and easy, but it requires the development of dedicated scripts on every device. Unfortunately, some of the devices available on the market do not have the ability to create scripts on devices. The availability is much better than in the case of OpenFlow protocol, but it may, in some cases, limit the implementation of the architecture.
- In the implementation process in an SMB production environment, the architecture was used in a process of securing the environment of critical IT system with a high risk of attacks. One of the implemented functionality was the ability to turn off the whole system (switches, routers, PCs, servers, stop traffic etc.) in case of an anomaly detection by calling one procedure in the supervisory application.
- By using this mechanism, it became possible to control not only the network devices but also servers, PCs, machines etc.
- The solution, for its simplicity, allows for quick and easy failure and errors detection.
- As is the case of the SDN architecture, it is necessary to introduce very restrictive control of modules of application responsible for network control.

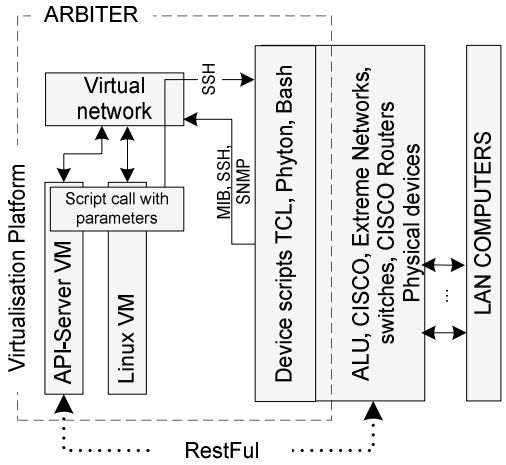

Figure 5. An arbiter architecture built on a controller with distributed scripts

The drawing also shows a new way to control devices using the RestFulAPI procedures. Some devices (Alcatel-Lucent) have provided such a communication interface, but at the moment it is a rather rare solution, and therefore its implementation potential is low.

3.4. Summary of Solution Selection

The table below summarizes all proposed architectures together with assessment criteria. Based on the analysis of the following solution, it becomes possible to select a particular software architecture to meet the requirements and capabilities of an SMB company. The following scales were used to assess the given architecture in the particular areas: +++ (very good), ++ (good), + (sufficiently), - (not accepted).

Table 1. Summary of proposed software architectures of arbiter implementation

Selection Criteria	SDN	Central Scripts	Distributed Scripts
Cost of implementation	+	+++	+++
Knowledge of programmers in area of computer networks	++	+	++
Ease of implementation	+/-	++	+++
Servers cost	+	+++	+++
Availability of architecture elements on network devices	+/-	+++	++
Scalability	+++	+	+
Openness of used standards and protocols	+++	++	++
Architecture modularity	+++	+++	+++

Popularity of used programing languages	++	++	++
Time needed for implementation	-/+	+++	++
System maintenance costs	++	+	+
Cost of implementation	+++	+	+

The results obtained show that the unambiguous answer, which solution is the best, is impossible. However, the results of the research conducted allow the managerial staff of the company to select the appropriate methods and means for implementing the software arbiter that manages the entire network environment. Of course, this decision depends on the business profile and the hardware and software limitations. Thanks to this approach, it will be possible to implement advanced management mechanisms for a company communication system while maintaining relatively low implementation and maintenance costs. Research has shown that the system can be implemented based on open standards and can be used in a heterogeneous hardware environment. The proposed architectures cannot only manage resources such as network devices and computers, but also any device that support the presented communication protocols. Preliminary work has shown that such communication methods (e.g. SNMP ssh) are implemented inter alia in numerical machine tools, industrial cameras, controllers, production lines. Such flexibility of the proposed solution allows it to be used in both IoT and Industry 4.0 systems.

4. Summary

The work presents methods and means of creating applications to control complex network environment. By the concept of complex network environment the authors understood both the complexity of the architecture in the area of topology, the used communication protocols (L2, L3, L3 and L4 of the ISO / OSI model) but also the ability to treat IT systems in terms of self-adaptive system and complex systems theory[17]. Thanks to the proposed software architectures of the arbiter implementation, it became possible to control the IT system centrally as a whole in an automatic manner. This approach fits in the concept of Service Oriented Architecture. As part of the re-search work, the proposed architectures were implemented in a lab environment and partly in the production environment of SMBs. The results of the work enable the selection of appropriate methods and means of implementation of the central arbiter managing the connection system in the environment of information systems dedicated to SMBs. The proposed solutions mostly use protocols and standards available on the market, which simplifies the implementation of the proposed architecture in a production environment. In further work, the authors wish to focus on the analysis of the use of a loop-back mechanism in terms of non-equilibrium thermodynamics and non-extensive statistics for controlling this class of systems.

References

[1] N. R. Bass, Component architecture in a network management, Bell Labs Technical Journal, vol. 8, issue 3, pp. 51-56, Alcatel-Lucent 2014

[2] B. Thornton, The applicability of network management systems in small businesses, Conference Anthology, IEEE 2013

[3] M.M Alani, Routing Protocols, Guide to Cisco Routers Configuration, Springer Briefs in Computer Science, pp. 5-21,Springer London, 2012

[4] D. Yuan, J. Jin, J. Grundy, Y. Yang, A framework for convergence of cloud services and Internet of things, Computer Supported Cooperative Work in Design (CSCWD), 2015 IEEE 19th International Conference on, IEEE 2015

[5] Ch. Yoon, Open IPTV convergence service creation and management using service delivery platform, Advanced Communication Technology (ICACT), 2015 17th International Conference on, IEEE, 2015

[6] V. M. Mondragon, V. García-Díaz, C. Porcel, R.G. Crespo, Adaptive contents for interactive TV guided by machine learning based on predictive sentiment analysis of data, Soft Computing A Fusion of Foundations, Methodologies and Applications vol. 21, issue 197, pp. 1-22, Springer Berlin Heidelberg 2017

[7] A.E. Tuchin, M. Sasabe, S. Kasahara, A simple algorithm of centralized flow management for data centers, Communications (APCC), 2016 22nd Asia-Pacific Conference on, IEEE 2016

[8] P. Civera, D. Del Corso, F. Maddaleno, Some Examples of Multiprocessor Buses, Multi-Microprocessor Systems for Real-Time Applications, Volume 2 of the series International Series on Microprocessor-Based Systems Engineering , pp. 165-224, Springer Netherlands 1985

[9] M. R. T. Tan, M. McLaren, N. P. Jouppi, Optical Interconnects for High-Performance Computing Systems, IEEE Micro vol. 33, issue 1, IEEE 2013

[10] OmniSwitch AOS Release 8 Switch Management Guide, Alcatel-Lucent 2014

[11] M. Bolanowski, A. Paszkiewicz: The Use of Statistical Signatures to Detect Anomalies in Computer Network. Analysis and Simulation of Electrical and Computer Systems, Lecture Notes in Electrical Engineering, vol. 324, pp. 251-260, Springer International Publishing 2015.

[12] Y. Wang, H. Wang, C. Han, B. Ge, M. Yu, Research on Information Fusion Method Based on sFlow and Netflow in Network Security Situation, Emerging Intelligent Computing Technology and Applications, International Conference on Intelligent Computing, International Conference on Intelligent Computing vol. 304, pp. 139-145, Springer-Verlag Berlin Heidelberg 2012

[13] http://www.rarr.rzeszow.pl/projekty/zakonczone-projekty/nauka-idzie-w-praktyke?PHPSESSID=d1cf1455c68d1ca525c50ebbb3cf7132

[14] http://www.rarr.rzeszow.pl/o_nas/aktualnosci/1398,rzeszowska-agencja-rozwoju-regionalnego-zaprasza-do-wziecia-udzialu-w-projekcie-transferencia.html

[15] C.R. Vasconcelos, R.C.M. Gomes, A.F.B.F. Costa, D.C.C. da Silva, Enabling high-level network programming: A northbound API for Software-Defined Networks, Information Networking (ICOIN), 2017 International Conference on, IEEE 2017

[16] Y. Gong, W. Huang, W. Wang, Y. Lei, A survey on software defined networking and its applications, Frontiers of Computer Science, vol. 9, issue 6, pp. 827–845, Higher Education Press, Springer 2015

[17] F. Grabowski, A. Paszkiewicz, M. Bolanowski: Wireless Networks Environment and Complex Networks. Analysis and Simulation of Electrical and Computer Systems, Lecture Notes in Electrical Engineering, vol. 324, pp. 261-270, Springer International Publishing 2015

Chapter 12

Cloud-based Vehicle Managing System

1. Introduction

The history of digital machines in automotive industry reaches back to the 1980s, when restrictive environmental standards forced manufacturers to more effective combustion of fuels in engines. To improve engine parameters mechanical control systems were used. Their main disadvantage was that dynamic regulation parameters of engine work was not possible. That was the reason for engineers to develop electronic controllers based on microchips. It was possible thanks to miniaturization and increase of computing power of microprocessor systems. First models of cars which used microprocessor systems to control the engine appeared in 1984-1985, but the beginning of car computerization started in 1990s when all the greatest manufacturers introduced Electronic Control Unit (ECU) in their cars.

Control system of average vehicle includes currently 25-30 microcontrollers and this number comes up to 70-80 in some vehicles. Intensive progress in electronics and information technology allowed more accurate control of physical processes, but also increased the number of monitoring sensors and microcontrollers that manage these processes. The greater number of system components may enlarge failure frequency. To overcome this, the system must be monitored. It also must inform user about possible defects and hazardous situations.

With the increased number of electronic components, there is a need to systemize and reduce quantity of information displayed on the car dashboard. For that reason every modern car is equipped with a main computer which informs the user about condition of the vehicle. The main computer monitors parameters and condition of car subassemblies. If they do not fulfill required or system detects malfunction, it warns the user.

Not every item of information is important to the driver, but data collected by car systems may improve comfort and safety of travellers and even may help to analyze conditions and situations on the road. Large amount of information which the car maintains often surpasses capabilities of the driver's perception. Additionally, only a few systems offer a function to collect the data and put it into a computing cloud for further examination of functioning of vehicles, or to analyze a fleet of vehicles and to visualize the results.

2. Cloud Computing in Vehicles

Term "cloud" was used in 1997 by Ramnath Chellap from Emory University with reference to a network which structure with servers necessary to process an information is hidden for final customer. Nowadays cloud computing is defined as usage of computer techniques connecting computing power of multiple machines in a network without revealing the physical infrastructure. One can say, that access to services, resources and information in the cloud is done by the user without any knowledge about processes and physical structure of computing system [1].

Every cloud computing system is built on a basic topology, which parts are clients and data centers with distributed servers (Fig. 1). Every part is an inherent element of the cloud structure and is acting a determined role. Datacenters are locations where machines conducting operations in the cloud are installed.

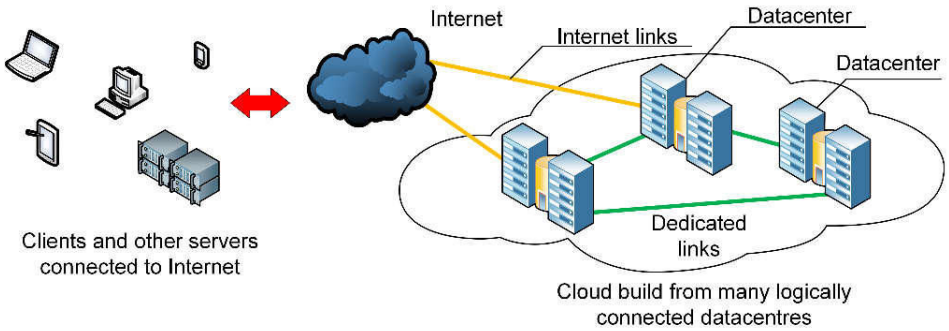

Figure 1. Basic cloud computing components

Usually there are located in a dedicated building, equipped with fast optical fibers links, redundant power supplies and an adequate cooling system. Datacenters keep data and its backup, maintain service servers available to the clients and conduct necessary calculations. It is worth to mention that datacenters are not the cloud itself, they are only physical elements of it. Currently the physical devices are virtualized which allows better scaling and optimize utilization of computing power. Multiple datacenters are often interconnected, making one logical structure, not dependent on geographical localization [2], [3].

Car industry is one of the domains in which cloud usage increases rapidly. The dedicated model of cloud computing in this field is called VCC (Vehicular Cloud Computing). It uses resources of the cloud for professional processing of car data and modern management of the vehicle by connecting it to a wireless network. This idea is currently intensively developed under the names *Connected Car* and *IoV* (*Internet of Vehicles*). The latter name was taken from popular conception of *IoT* (*Internet of Things*).

Table 1. Services and features offered by different car companies

		AUDI	BMW	Mercedes	Tesla	Ford	Nissan	Opel	Peugeot	Renault	Toyota
Safety	remote diagnostics		•	•	•			•	•	•	
	anti-theft assistnance				•			•			
	speed monitoring			•	•			•	•		
	emergency call	•	•	•	•	•	•	•	•	•	•
	assistance	•	•	•	•			•	•	•	
infotainment services	web browser		•	•	•						
	Wifi-Hotspot	•	•	•	•			•	•		
	news feed	•	•	•	•	•	•			•	
	multimedia streaming	•	•	•	•	•	•	•	•		•
	e-mail	•	•	•	•		•	•	•	•	•
	social media	•	•	•	•		•		•	•	
	App store	•	•	•	•	•	•	•	•	•	
navigation services	Street View	•			•						•
	Road traffic	•	•	•	•	•	•	•	•	•	•
	route planning	•		•	•		•	•		•	•
	Parking lot finder	•	•	•	•	•			•	•	•
exploatation expences	driver style monitoring			•			•			•	
	real time fuel prices	•	•	•			•		•	•	•
	electric car charging asstistance		•		•		•		•	•	•
	next service calculation	•	•	•	•	•		•	•	•	
convenience s	Automatic toll collection										
	Customer service		•	•	•			•			
	Remote management services	•	•	•	•	•	•	•	•	•	•
	Personal web platform	•	•	•		•		•	•		

List of services offered by manufacturers is presented in Tab. 1. In spite of several years long existence of these solutions on the market and development of advanced navigation functions such as information about traffic, free parking areas, implementation of sophisticated multimedia systems, there is still a lack of functions improving safety on the road, such as remote diagnostics or monitoring parameters of the vehicle and style of driving.

There are dedicated IT solutions for cars such as Android Automotive (Google) or QNX Car Platform (Blackberry). However, in spite of increasing number of vehicles with communication functions, usefulness of gathered information is rather small. One of the main causes is lack of standardization of communication protocols between cars of various brands and vendors of cloud services. This makes difficulties in implementation of such services as navigation supported by movement of the vehicles or reporting about road collisions and traffic congestion. In addition, many monitoring systems [4], [5], [6] are designed as specialized services which utilize private clouds and are incompatible with each other.

This is the main reason for an idea of a low-cost universal device that would continuously collect car and engine parameters and communicates with a public cloud application to store and process the data. Here we present design and implementation of a prototype of such system.

3. Concept of Vehicle Management System

The presented system has been designed to handle read-out of vehicle data and transfer it from a car to a computing cloud. Main goal of the project was to design a low-cost system which uses public cloud and can be expanded to more than one vehicle or even a fleet of cars. The system prototype uses popular Raspberry Pi board with Linux operating system to prove that such service does not need expensive hardware to operate. Similarly, worldwide Azure public cloud has been utilized to show that low-cost solution is possible to implement data services. The block scheme of the system is shown in Fig. 2.

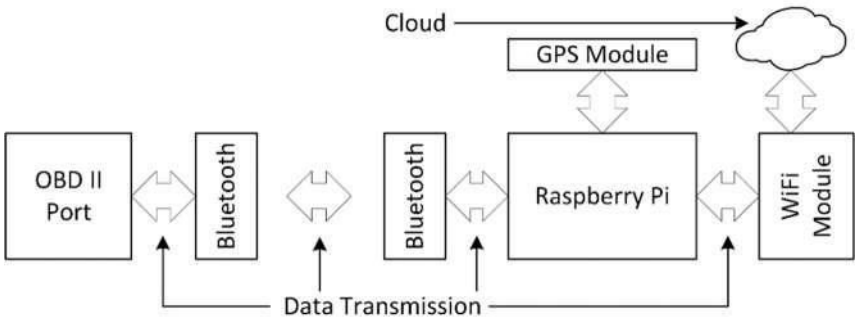

Figure 2. Block scheme of system designed for data acquisition

This is the main reason for an idea of a low-cost universal device that would continuously collect car and engine parameters and communicates with a public cloud application to store and process the data. Here we present design and implementation of a prototype of such system.

The user has the ability to interact with the system with a touch screen, however direct interaction with the reading and logging software is not needed by the driver. Configuration of the software is made in service text mode when the system does not operate during ride. During initial configuration an unique identifier (*CarId*) is set up for each vehicle in a fleet. Data obtained by RPi is transmitted via the internet to a database server running on the Azure cloud. In addition, the collected data is available from anywhere using a web browser, what makes the project widely accessible and eliminates the need to build a custom data viewing application for particular operating systems.

From technical view, the presented setup works independently from car systems. Also, the data is transferred one way, i.e. from RPi to the cloud. We assume here a protected channel to keep the data secure. If more integration with the car is considered or return commands which would influence car behavior are introduced, safety standards such as [7] should be taken into account.

4. Software Design

Software architecture of the vehicle management system involves three communication components. The first one is used to read car data by the executive unit (RPi) via OBD II diagnostic protocol [8]. The second one collects data from GPS module. The third component sends data from the executive unit in the car to the Azure Cloud. From this perspective, the Raspberry Pi based executive unit can be seen as a data exchange hub. It runs multitasking operating system called Raspbian, being a variation of Linux [9], [10].

Raspberry Pi UART interface has been utilized to connect the GPS module. The module sends GPS data such as latitude, longitude and time using NMEA protocol. U-blox NEO-6 GPS module has been used in the prototype of the system. The chip integrates Automotive Dead Reckoning (ADR) to improve position accuracy by using a tightly coupled Kalman filter [11]. This feature allows the system to estimate position of a vehicle when satellite signals are not available, for example in a tunnel [12].

To handle GPS data coming from the module three software components have been used, namely: *gpsd*, *gpds-clients* and *python-gps*. They allow to read, interpret, analyze and display GPS data. Fig. 3 shows a sample output of a tool called CGPS.

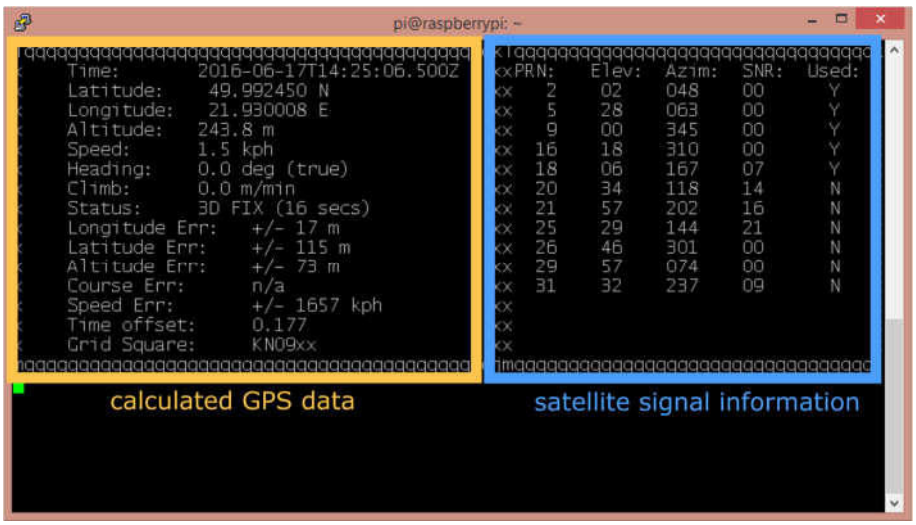

Figure 3. Display of position data obtained from GPS module

In case of OBD II communications a Bluetooth interface has been used. The tools *bluez* and *blueman* have been utilized to manage the connection. Communication software has been written in Python language using libraries for Bluetooth and OBD (*obdython*). It establishes communications with the car, sends OBD commands and reads vehicle data (Fig.4) [13], [13].

The collected data is sent to the cloud server via the internet. The server (called *cloudcar*) has been created in Azure cloud system. Its main role is to provide database services. For this purpose cloud-based MS SQL Server database has been set up. Structure

of the main table in the database (*carlog*) where the collected data is stored, is presented in Tab.2.

```
pi@raspberrypi:~/pyobd-pi $  python ./obd_recorder.py
Logging item: Engine RPM
Logging item: Vehicle Speed
Logging item: Throttle Position
Logging item: Calc Load Value
Logging item: Fuel System Stat
['/dev/rfcomm0']
Opening interface (serial port)
Interface successfully /dev/rfcomm0 opened
Connecting to ECU...
atz response:atzELM327 v2.1
ate0 response:ate0OK
0100 response:BUS INIT: ...0K41 00 BE 1F E8 11
Connected to /dev/rfcomm0
Logging started
```

Figure 4. Logging vehicle data from OBD module

Table 2. Structure of the table *carlog*

ID	CarID	Date	Time	Latitude	Longitude	Speed	RPM	Throttle	Collant temp.	Air temp.

The first column *ID* is the primary key of the log entry table. It is followed by *CarID*, so the table can hold information coming from multiple cars in a fleet, each one with an unique identifier. The next two columns *Date*, *Time* contain timestamp of the data. It is worth to mention that the information is the actual timestamp of when the data is obtained by the executive unit (RPi) and is not related to the time when the data is inserted into the database by the server. This way delays introduced by internet communications with the cloud do not cause a problem. The columns *Latitude* and *Longitude* store GPS data or the text "NO DATA" when it is not available. *Speed* and the following columns hold data obtained from the vehicle via OBD II interface. The speed is a non-negative integer even when the car moves in reverse. *RPM* denotes engine revolutions per minute while *Throttle* is a percentage of the throttle position. The values in the last two columns (*Coolant temp., Air temp.*) are also obtained from the car via OBD II.

The table *carlog* is populated upon internet requests from the RPi executive unit located in the car using a Python program. The simplified code for generating the requests is shown in Listing 1. We will briefly describe it now. The first lines are used to import required Python libraries and modules, such as *math*, *time* or *datetime*. *Db* is used for database connections. *InsertQuery* for data insertion into the database, while *gps* and *obdlibrary* contain functions for handling GPS and OBD interfaces. The following line establishes connection with the Azure cloud giving the server address, username and password, and database name as parameters. Then GPS and Bluetooth communications are opened. The main portion sets a set of variables by reading appropriate values from the system (*date*, *time*) and the OBD port (*speed*, *rpm*, *throttle,* etc.). The GPS latitude obtained from the module is then checked for validity. If it is valid, all the values are used by *InsertQuery* to form a proper SQL command for the database server. The command is then passed to be executed by the remote Azure cloud server. This way the date are collected in the cloud for further advanced processing and display.

Listing 1. Python code of sending car data to the cloud.

```
import time
import math
import datetime
from db import Db
from insert_query import InsertQuery
from gps import *
from obdlibrary import Device, OBDPort

azure = Db('cloudcarraspberry.database.windows.net',
'user@cloudcarraspberry', 'password', 'cloudcar') #open cloud connection
gpsd = gps(mode=WATCH_ENABLE) #open communications with gps module
dev = Device(Device.types['bluetooth'],
bluetooth_mac="00:1D:B3:00:09:29", bluetooth_channel=1)
port = OBDPort(dev)
time.sleep(0.1) # pause - waiting for devices initialization
port.connect() # opening connection
time.sleep(0.1)
port.ready()# checking if port is ready

date = datetime.datetime.now() # getting actual date
time = datetime.datetime.now().time() # getting actual time
speed = port.sensor('speed')[1] # getting actual speed
rpm = port.sensor('rpm')[1] # getting RPMs
throttle = port.sensor('throttle_pos')[1] # getting throttle position
coolant_temp = port.sensor('temp')[1] # getting coolant temperature
air_temp = port.sensor('intake_air_temp')[1] # getting intake air
temperature
gpsd.next()
if gpsd.fix.latitude != 0 and not math.isnan(gpsd.fix.latitude) : # if
position data is obtained then send obtained data to database
        sql_query = InsertQuery({ 'Date': str(date),'Time': str(time),
'Latitude' : str(gpsd.fix.latitude), 'Longitude' :
str(gpsd.fix.longitude), 'Speed':str(predkosc),'RPM':str(obroty),
'Coolant_temp':str(coolant_temp), 'Air_temp':str(air_temp)}) # method for
sending values and inserting them in correct table fields
        print azure.execute(sql_query); # execute data insertion
azure.close() # close connection
```

5. Data Visualization

The important element of the system is visualization of the collected data by a web browser. It has a form of a website with a number of interactive charts with data collected from the vehicle as shown in Fig. 5.

The upper part contains charts for speed, acceleration and RPM in function of time. As seen, the three variables are combined in a single chart. Since the database may contain a huge number of records, it is necessary to choose a period of time for data visualization. By moving a mouse one can observe the actual value at the indicated point. The charts are scaled automatically and may be exported to a file. The other charts in Fig. 5 may be useful for analyzing the data. The central part is occupied by a speed chart in function of acceleration and a driven distance chart. The lower part contain two point charts: rotation speed in function of time, and acceleration in function of rotation speed. The pie chart at

the lower right shows the time spent in defined ranges of engine rotation speed. The charts allow to identify driving style of a driver, including economy and dynamics. For example, high value of speed and low value of rotations may indicate less fuel consumption.

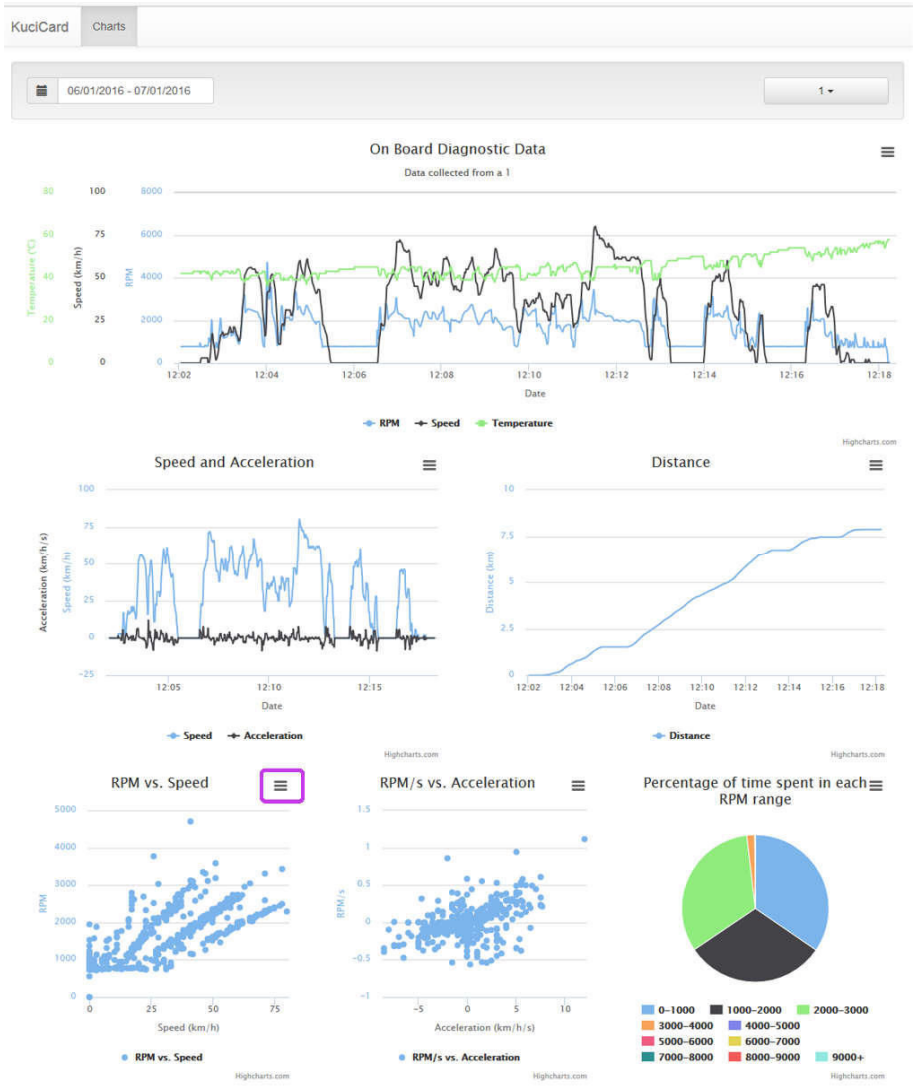

Figure 5. Visual presentation of vehicle data on a web page

The visualization website has been built a service running on Microsoft Azure. Its primary task is to download data from SQL server and display it in the form of the interactive charts. Apart from the service, the following techniques have been used to create the web interface:

- *Bootstrap* – CSS framework for creating scalable pages working on different platforms, such as personal computer and mobile devices.
- *jQuery* – programming library for JavaScript. It allows to expand functionality of websites.
- *Moment.js* – JavaScript library for managing time/date in many formats.
- *Highcharts* – JavaScript library for interactive charts.
- *Bootstrap date range picker* – script which allows to choose specified time period.

In the final phase of the project, it was important to test the operation of the system in the real environment and thus in the vehicle moving along the way.
Verification of the received data consisted of reading the position from the GPS module, velocity and other diagnostic data with constant time interval. Then acquired data were compared to those written in cloud. Data were acquired on the way from Rzeszow in Poland to Ingelheim am Rheim in Germany.

The charts presented in Fig. 5 show data for a single car selected by the user. In case of a fleet of cars, one can examine data for multiple cars. This may be useful to compare driving styles or to get some statistics such as overall condition of the cars, driving economy for the whole fleet etc.

6. Conclusions

The goal of the work was to design and implement a low-cost vehicle management system that uses cloud computing to store data about vehicles and driving parameters. The system involves a IoV device mounted in a vehicle to acquire data from internal bus of the car and to send it to the cloud database. The core of the solution is RPi board with Bluetooth and Wi-Fi modules and a touch screen. RPi is connected to the diagnostic interface OBD II.

The collected read-outs are placed in the remote database on Microsoft Azure cloud environment. The database content may be used for further research. The database may be used to analyze habits of the driver, driving style and other parameters. Web visualization with interactive charts can be used for the analysis. It is planned to develop algorithms which may assist the driver during travelling, alarming him about dangers, bad habits, condition of the car or provide other personalized information. It is also possible to combine this with information obtained from web services proving actual situation on the roads, weather conditions and traffic congestion to improve safety and comfort of the travel. Since the database collects data from multiple vehicles in a fleet, it is possible to obtain generic information about traffic, cars and drivers. This may be useful for car manufacturers or transportation companies to improve car fleets and autonomous cars.

Low cost is a special factor which has been taken into account while designing. As the result, the proposed system has been built with the usage of affordable and universal parts. A public cloud service has also been utilized.
In the next stage of the project, the authors plan to extend the system with an RFID (Radio-frequency identification) reader. This will allow to integrate it with a traffic management system built with the usage of RFID transponders (Fig. 6). Such

transponders can store specific information about road type, maximum allowed speed, actions prohibited on particular road and many more. Thus, it will be possible to collect and analyze information not only about vehicle itself but also about its environment what in the future may introduce a new quality in this type of systems.

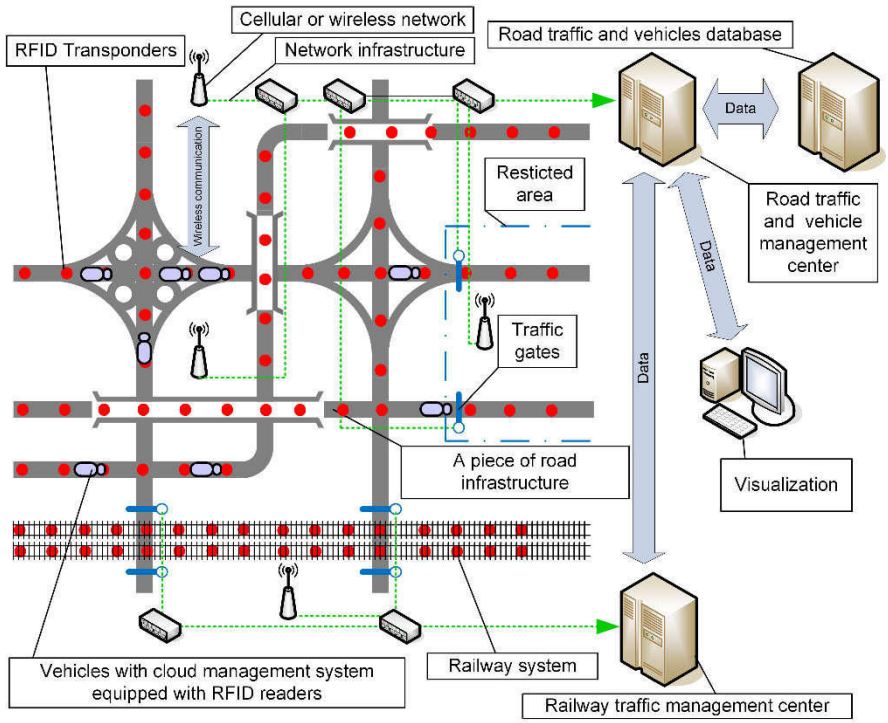

Figure 6. Planned architecture of cloud management system with RFID technology

References

[1] B. Sosinsky, Cloud Computing Bible, Wiley Publishing Inc., 2011
[2] T.A. Velte, T.J. Velte, R. Elsenpeter, Cloud Computing: A Practical Approach, McGraw Hill, 2010
[3] D. Rountree, I. Castrillo, H. Jiang, The Basics of Cloud Computing, Syngress, an imprint of Elsevier, 2014
[4] N. Sharma, N. Chauhan, N. Chand, Smart logistics vehicle management system based on internet of vehicles, 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC), 22-24 Dec. 2016, Waknaghat, India, ISBN: 978-1-5090-3669-1
[5] L. Wu, F. Qiao, J. Lu, Design and implementation of vehicle management system based on the ubiquitous network, 2013 IEEE 4th International Conference on Electronics Information and Emergency Communication (ICEIEC), 15-17 Nov. 2013, Beijing, China, ISBN: 978-1-4673-4933-8
[6] Y. Leng, L. Zhao, Novel design of intelligent internet-of-vehicles management system based on cloud-computing and Internet-of-Things, 2011 International Conference on Electronic and Mechanical Engineering and Information Technology (EMEIT), 12-14 Aug. 2011, Harbin, China, ISBN: 978-1-61284-088-8
[7] ISO 26262 Road vehicles - Functional safety, 2011, 2012
[8] A. Santini, OBD-II: Functions, Monitors and Diagnostic Techniques, Cengage Learning, 2010
[9] D. Molloy, Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux, Wiley, 2016

[10] W. Gay, Mastering the Raspberry Pi, Apress, 2014
[11] R. van der Merwe, E.A. Wan, Sigma-point Kalman filters for Integrated Navigation. Proc. 60th Ann. Conf. of the Inst. Navigation, Dayton, OH, June 7-9, pp.641-654, 2004
[12] E. Favey, A. Somieski, C. Hollenstein, M. Ammann, C. Fenger, Design How-To: Dead reckoning fills-in GPS navigation gap. EE Times 18th August 2011
[13] T. Zhang, L. Delgrossi, Vehicle Safety Communications: Protocols, Security, and Privacy 1st Edition, Wiley, 2012
[14] M. di Natale, H. Zeng, P. Giusto, A. Ghosal, Understanding and Using the Controller Area Network Communication Protocol, Springer, 2012

Authors and affiliations

Piotr KOSIUCZENKO – Scientific Editor
Institute of Computer and Information Systems, Faculty of Cybernetics, Military University of Technology, Warszawa, Poland
piotr.kosiuczenko@wat.edu.pl

Lech MADEYSKI – Scientific Editor
Department of Software Engineering, Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wrocław, Poland
lech.madeyski@pwr.edu.pl

Mirosław OCHODEK – Scientific Editor
Institute of Computing Science, Poznań University of Technology, Poznań, Poland
miroslaw.ochodek@cs.put.poznan.pl

Andrzej PASZKIEWICZ – Scientific Editor
Department of Complex Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
andrzejp@prz.edu.pl

Anna DEREZIŃSKA – Chapter 1
Institute of Computer Science, Warsaw University of Technology, Warszawa, Poland,
a.derezinska@ii.pw.edu.pl

Wiktor NOWAKOWSKI – Chapter 2
Warsaw University of Technology, Warszawa, Poland
wiktor.nowakowski@ee.pw.edu.pl

Kamil RYBIŃSKI – Chapter 2
Warsaw University of Technology, Warszawa, Poland
rybinskk@iem.pw.edu.pl

Michał ŚMIAŁEK – Chapter 2
Warsaw University of Technology, Warszawa, Poland
michal.smialek@ee.pw.edu.pl

Andrzej ZALEWSKI – Chapter 3
Institute of Control and Computation Engineering, Warsaw University of Technology, Warszawa, Poland
a.zalewski@ia.pw.edu.pl

Andrzej RATKOWSKI – Chapter 3
Institute of Control and Computation Engineering, Warsaw University of Technology, Warszawa, Poland
a.ratkowski@elka.pw.edu.pl

Małgorzata PURWIN – Chapter 3
Institute of Control and Computation Engineering, Warsaw University of Technology, Warszawa, Poland

Marcin DASZUTA – Chapter 4
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
173059@edu.p.lodz.pl

Dominik SZAJERMAN – Chapter 4
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
dominik.szajerman@p.lodz.pl

Piotr NAPIERALSKI – Chapter 4
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
piotr.napieralski@p.lodz.pl

Michał WROŃSKI – Chapter 5
Department of Complex Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
mwronski@prz.edu.pl

Grzegorz KOCHAŃSKI – Chapter 6
Smart4Aviation, Gdansk Science & Technology Park, Gdańsk, Poland
grzegorz.kochanski@smart4aviation.aero

Aneta PONISZEWSKA-MARAŃDA – Chapter 7
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
aneta.poniszewska-maranda@p.lodz.pl

Joanna PAWELSKA – Chapter 7
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
190154@edu.p.lodz.pl

Aneta MAJCHRZYCKA – Chapter 7
Institute of Information Technology, Lodz University of Technology, Łódź, Poland
anneta.michalska@gmail.com

Radosław MARKIEWICZ – Chapter 8
Monterail.com sp. z o.o., Wrocław, Poland
radomark@gmail.com

Ziemowit NOWAK – Chapter 8
Faculty of Computer Science and Management, Wrocław University of Science and Technology, Wrocław, Poland
ziemowit.nowak@pwr.edu.pl

Andrzej TURNAU – Chapter 9
Department of Automatics and Biomedical Engineering, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, AGH University of Science and Technology in Kraków, Poland
atu@agh.edu.pl

Wojciech ZWONARZ – Chapter 9
Department of Automatics and Biomedical Engineering, Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, AGH University of Science and Technology in Kraków, Poland
wzwonarz@gmail.com

Wiktor B. DASZCZUK – Chapter 10
Institute of Computer Science, Warsaw University of Technology, Warszawa, Poland
wbd@ii.pw.edu.pl

Maciej BIELECKI – Chapter 10
Warsaw University of Technology, Warszawa, Poland
M.Bielecki.1@stud.elka.pw.edu.pl

Jan MICHALSKI – Chapter10
Faculty of Electronics and Information Technology, Warsaw University of Technology, Warszawa, Poland
J.Michalski@stud.elka.pw.edu.pl

Marek BOLANOWSKI – Chapter 11
Department of Complex Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
marekb@prz.edu.pl

Andrzej PASZKIEWICZ – Chapter 11
Department of Complex Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
andrzejp@prz.edu.pl

Mateusz KUT – Chapter 12
Boehringer-Ingelheim, Ingelheim am Rhein, Germany
mateusz.kut@boehringer-ingelheim.com

Bartosz PAWŁOWICZ – Chapter 12
Department of Electronic and Communication Systems, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
barpaw@prz.edu.pl

Bartosz TRYBUS – Chapter 12
Department of Computer and Control Engineering, Faculty of Electrical and Computer Engineering, Rzeszow University of Technology, Rzeszów, Poland
btrybus@prz.edu.pl

Scientific Council
of the Polish Information Processing Society
Solec St. 38/103
00-394 Warszawa, Poland
ISBN 978-83-946253-6-8