e-Informatica

software engineering journal

2013 volume 7 issue 1

e-Informatica

e-Informatica

software engineering journal

2013 volume 7 issue 1

e-Informatica

Wrocław University of Technology

Editors

Zbigniew Huzar (*Zbigniew.Huzar@pwr.wroc.pl*)
Lech Madeyski (*Lech.Madeyski@pwr.wroc.pl, http://madeyski.e-informatyka.pl/*)

Institute of Informatics
Wrocław University of Technology, 50-370 Wrocław, Poland

e-Informatica Software Engineering Journal
http://www.e-informatyka.pl/wiki/e-Informatica/

Wojciech Thomas (Editorial Office Manager).

Typeset by Wojciech Myszka with the LaTeX 2$_\varepsilon$ Documentation Preparation System

Printed in the camera ready form

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław
http://www.oficyna.pwr.wroc.pl;
e-mail: oficwyd@pwr.wroc.pl; zamawianie.ksiazek@pwr.wroc.pl

ISSN 1897-7979

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Order No. xxxx/2012.

Editorial Board

Contents

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 7–14, DOI 10.5277/e-Inf130101

Comparison of MDA and DSM Technologies for the REA Ontology Model Creation

Zdeněk Meliš*, Jaroslav Žáček*, František Huňka*
*Faculty of Science, University of Ostrava
zdenek.melis@osu.cz, jaroslav.zacek@osu.cz, frantisek.hunka@osu.cz

Abstract

Using the ontology in information systems means an improvement of the possibility of solving tasks based on domain knowledge. The quality design of the ontological model is base of well-functioning system. In this area the most commonly used technology is MDA (Model driven development), which provides solid base for modeling language's metamodel definition. This article aims to compare this technology with a new approach to visual modeling – the DSM (Domain-specific modeling) technology. Using the narrow focus on specific domain and full code generation the DSM allows easy and rapid development of the ontological system. The REA (Resource–Events–Agents) ontology, intended for business processes modeling was used for comparison of these technologies.

1. Introduction

Currently the interest in developing information systems based on ontology is growing. With an increasing complexity of such systems, requirements for modeling principles with the high level of an abstraction to be able to transform created models to basic structures of an information system are growing [1]. The most commonly used technology is MDA that specifies functional details by a progressive model transformation. The DSM technology operates on a different principle. It allows creating models in a domain language and performs model validation and verification and full code generation.

First paragraphs describe the definition of the REA ontology and compared technologies – MDA and DSM, their principles and characteristics. The second part of the article deals with their comparison.

2. The REA ontology

According to [2] the ontology is a specification of a conceptualization. It is study of things that exists or can exist in explicit domain. A conceptualization means an abstraction and simplified view of the world. A specification means formal and declarative representation [3]. The ontology provides number of resources for intelligent systems, knowledge representation and knowledge engineering processes [4].

The REA is a concept for designing enterprise infrastructures based on ownership and their exchange. It is based on a concept of economic exchanges and conversions that increases company's value. According to [5] the ontology basis contains 5 parts, as you can see on Figure 1:

– Economic resource – elementary economic resource that company wants to plan, monitor and control. This resource can include raw material, money, work, labor, etc.

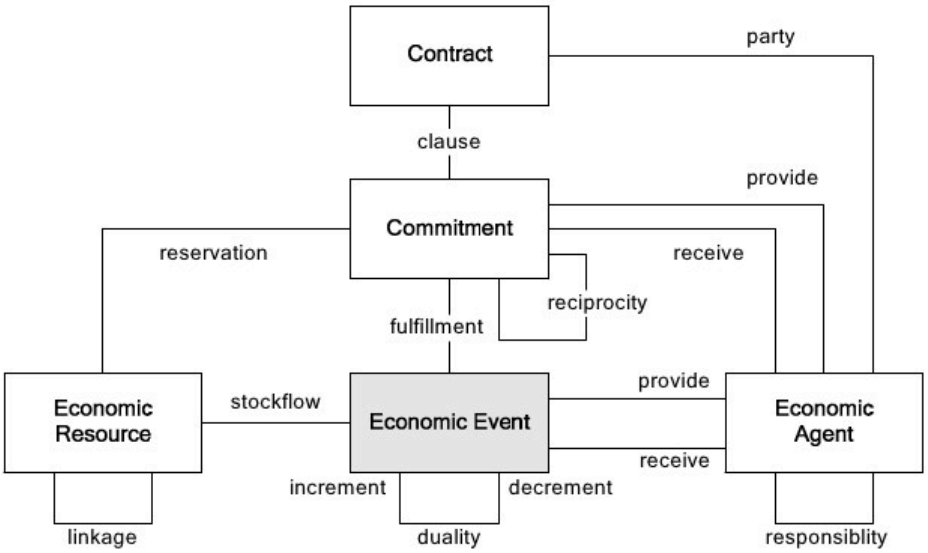

Figure 1. Fundamental REA concepts [5]

– Economic agents – an individual, group or company that controls economic resource and cooperate with other economic agents. An example can be customer, seller, employer, company etc.
– Economic event – represents economic change of resource (an increment or a decrement). This change can be immediate or long-term. An example can be work, using of services, renting, etc.
– Commitment – promise or obligation to perform an economic event in future.
– Contract – a set of commitments and rules (e.g. what happen when the commitment is not fulfilled)

Figure 1 shows fundamental model of the main concepts of REA ontology and links between them. One of the most important links is duality that answers us the question why an economic event occurs.

2.1. Levels distribution of the REA model

We can extend a basic view level by adding other entities to obtain an additional functionality. The REA ontology can be divided according level of an abstraction [6]:

– Value system level – the highest level of an abstraction. It describes the resources flow between the enterprise and its business partners.
– Value chain level – it divides an enterprise into strategically important activities. The enterprise gains a competitive advantage by doing these activities cheaper and better than competitors [7]. The view of this level describes resources flow between individual business processes.
– Model level – models describes transformation one economic resource into other, more valuable for enterprise. Figure 2 shows some concepts of model level and their links.
– Task specification level – the lowest level of an abstraction, it is an application of the model and contains raw company's data.

Generally the REA model level is divided into 2 levels according functionality [5]:

– Operational level is the basic skeleton of model. It describes events that already happened. Basic semantic abstractions of operational level are exchange, conversation and the value chain. Exchange and conversation increase an enterprise value and the value chain describes connection of various REA models into chain directly or indirectly contributing creation of desirable features of the final product or service. That final product can be exchanged for a more valuable resource with other economic agents.

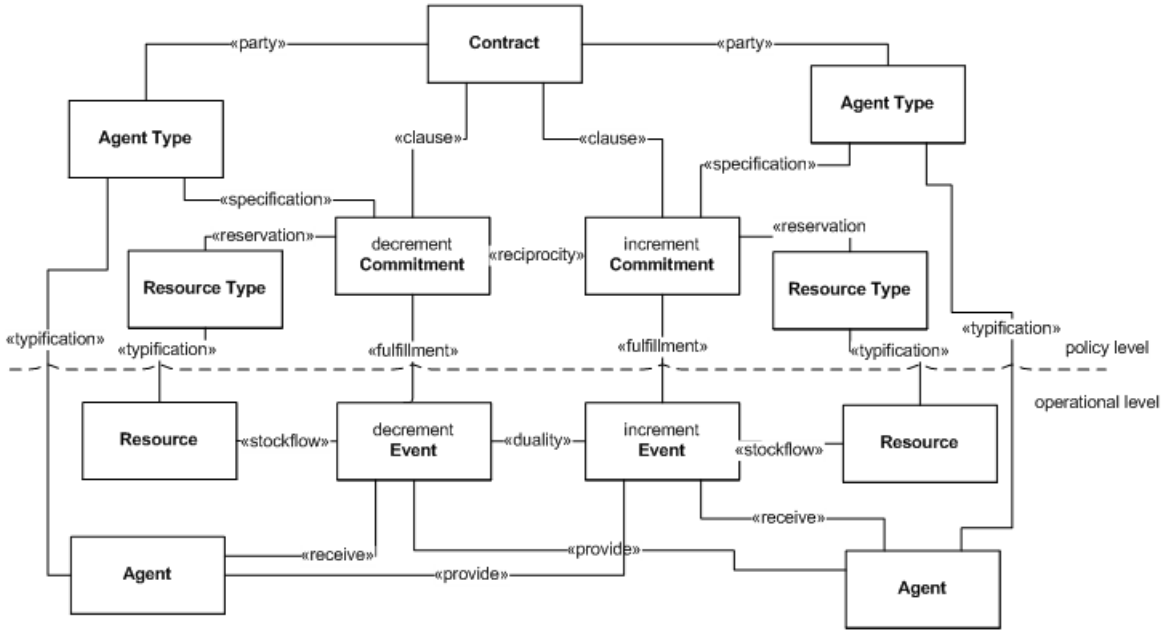

Figure 2. Example of business process model

– Policy level is an extension of operational level. It contains semantic abstractions describing what could, should or shouldn't happen such as group, type, contract, etc.

Figure 2 shows an example of business process model described by REA model level. It contains basic concepts of operational level and some semantic abstractions of the policy level.

2.2. Usage of the REA framework

Traditional business processes modeling approaches, such as flow chart, data model, use case, IDEF0, etc. use general concepts that are inappropriate because of low model specificity and therefore they cannot detect economical errors and make automation. The REA ontology uses specified concepts increasing amount model information while maintaining the model simplicity.

The REA model has internal rules for verifying the model consistency thus prevents creating of incorrect links. The result of this verification is the model, which is responding to an answer why the enterprise performs some activity and hence why economic events happened. This is a significant difference and a big advantage of the REA ontology over other traditional model solutions.

Another feature of the REA ontology is simplicity and understandability of models for ordinary users working with them. The model is also precise enough for automation [5].

3. Model Driven Architecture

MDA is a specification of OMG consortium (Object managment group) used for model driven software development. Model is simplified view of reality that defines formal set of elements to describe an objective and a purpose of development.

The reason for creating the MDA standard was an effort to increase the level of abstraction. Anytime in the history increasing level of abstraction led to increasing productivity.

3.1. MDA architecture

MDA is based on four-layer architecture (see Figure 3). The highest layer M3 is meta-metamodel. It is an abstract language and a framework for defining, specifying, designing and managing technologically independent metamodels and

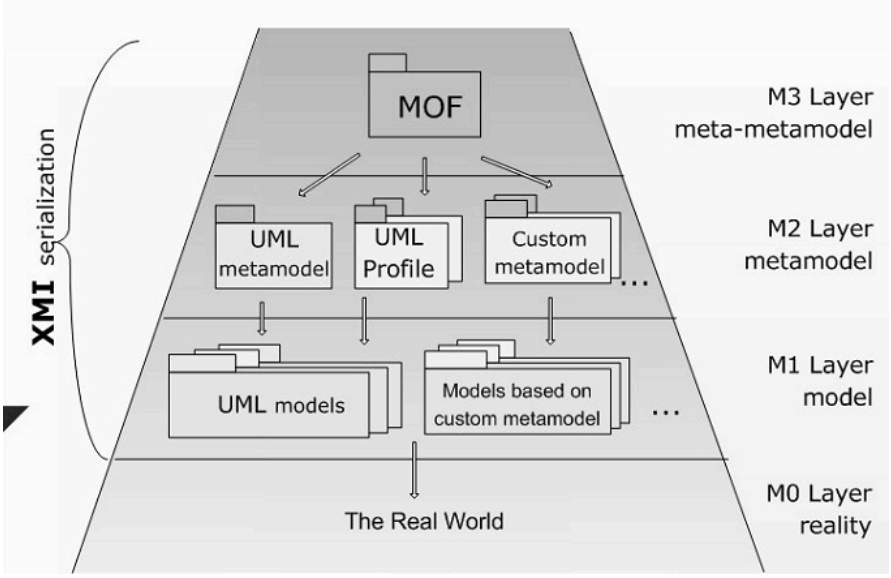

Figure 3. MDA architecture [3]

serves as the base for defining modeling language. The most commonly used language is Meta-Object Facility (MOF).

The second layer M2 contains all metamodels and specifications defined by top layer. In this layer there is usually defined UML language and its concepts (e.g. classes, associations, ...). The third layer M1 includes real world elements represented by metamodel concepts. The lowest layer M0 contains things of real world modeled in M1. This layer can include instances of concepts defined in M1 layer, or specific or abstract instances of real world.

3.2. MDA levels of abstraction

MDA defines 4 levels of an abstraction:

- CIM (the computational–independent model) – contains basic domain model with low level structure details. It models basic system requirements and basic business processes.
- PIM (the platform-independent model) – it is a model containing more details than CIM, but it is still free from technological details. Theoretically the PIM is assumed to be executed on a technologically independent virtual machine. This model describes the most of a system behavior.
- PSM (the platform-specific model) – in the PSM details such as the code structure for selected platform are generated, and constructs of the final language and details enabling code generation are completed (usually automatically).
- Code – the source code can be understood as the model of concrete realization on the platform.

3.3. Ontology infrastructure based on the MDA

The OMG Company tried to create an ontology system based on MDA to approach ontology systems to common programmers.

Figure 4 shows the ontology-based infrastructure made by MDA. The highest layer M3 contains basic MOF defining meta-metamodel. The M2 layer contains basic UML ontological profile describing basic modeling language's constructs and Ontology Definition Metamodel (ODM), which includes basic ontology concepts. This language is based on OWL, which is the result of the evolution of ontology languages. This layer forms a logical layer of the semantic web and allows mapping from ODM to OWL using XMI and XSL format (based on XML). The M1 layer contains general models based on higher layer

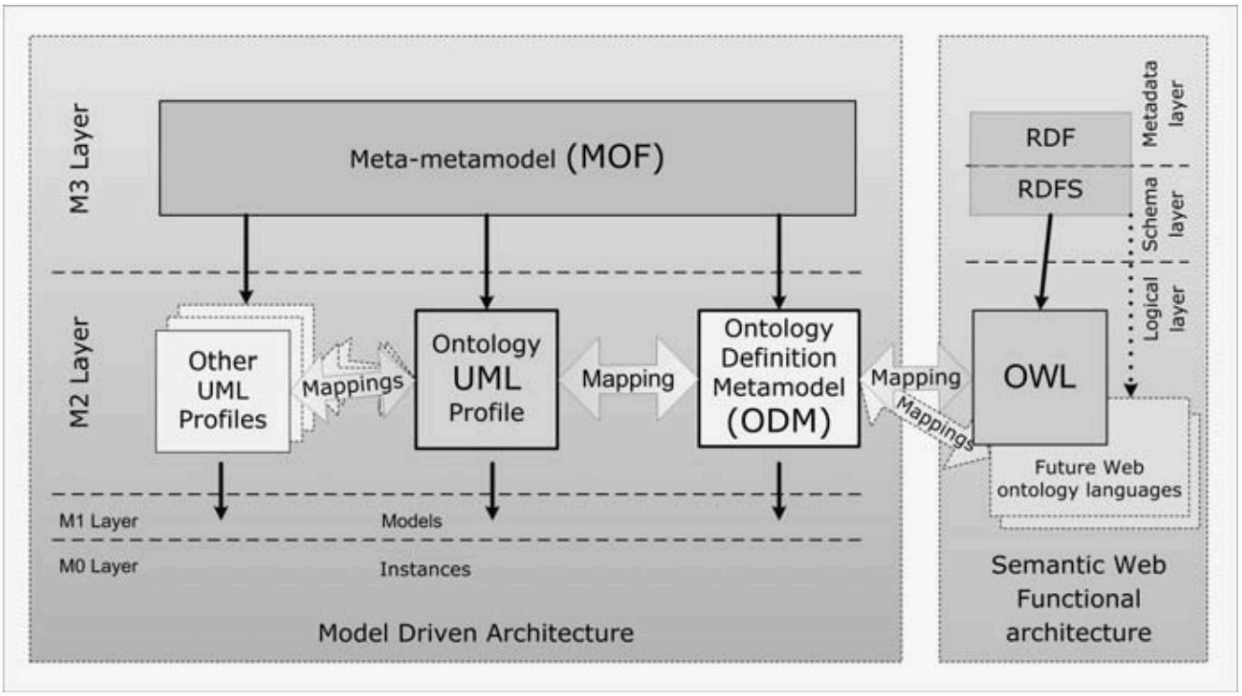

Figure 4. MDA based ontology infrastructure [3]

metamodels and M0 contains model instances and raw data.

MDA technology does not distinguish the type of the ontology - the general transformation process can be applied to any ontology, including the REA. The highest layer is MOF, M2 layer includes the REA ontology UML profile and M1 layer consists individual models.

4. Domain-specific modeling

Domain-specific modeling (DSM) is a software engineering methodology for software development [8]. The main focus of DSM is automated development in one narrow specific domain. The advantage of narrow focus is possibility to use domain terms in language including its graphical notification. DSM increases the level of an abstraction above code concepts eliminating the need for mapping of domain concepts into another language and reduces mapping errors [9].

4.1. DSM architecture

The DSM architecture contains 3 parts: the language, the generator and the domain framework. The language comes directly from a problem domain and provides an abstraction for solving domain problems. Individual domain concepts can form language objects, links, properties or submodels. The language itself has two parts. The syntax specifies language concept structure and uses a grammar to perform model validation and verification at the language level. The semantic defines the meaning of individual elements.

The generator performs model transformation to pre-defined structure, usually the source code which is complete and immediately executable without any modifications. The generator can produce documentations, metrics, statistics, prototypes, tests and more.

The lowest layer is the domain framework that performs multiple functions. It reduces complexity of code, eliminates duplicity in code, de-

fines an interface for generator, provides integration with existing systems, etc.

4.2. Ontology infrastructure based on the DSM

The figure 5 shows detailed DSM mapping of the ontology system. The vertical part of diagram shows the hierarchical ontological structure where Upper Ontology defines abstract layer of concepts usable in all domains, Core Ontology defines concepts usable in similar domain, Domain Ontology defines specific domain concepts and Application Ontology defines concepts that are modified for DSL mapping [10].

Unlike MDA, the DSM technology is directly dependent on the modeled ontology. Due the strict dependence on the domain, procedure of creation architectural layers of modeling system is different for each ontology.

The language of the REA ontology is defined by the UML profile and contains basic entities, relationships rules that allow validation and verification of the model. The target platform is hidden from user and no other action is required for creating a functional model so the system is able to work with an instance level of model. The core of such system is the generator, performing a translation of the model into the target platform code. Making of such generator is very difficult, expensive and time-consuming and requires domain expert participation. Once the generator is created, models creation is very easy even for nontechnical people.

5. Comparing MDA and DSM principles

5.1. MDA result

The MDA is based on incremental transformations that specify model details through all abstraction levels. This approach allows transformation of any REA ontology model independently on used entities, because transformation process is not affected by the applied semantic abstractions. The MDA structure allows easy transformation of the REA ontology language to any other language (e. g. to OWL language for semantic web support) without modification of the modeling system's core. Very helpful function may be the learning ability - the model learn transformation details set by a model creator and reuse them at the next transformation of the model with the same combination of used entities.

The big problem of this approach is that a model creator must have programming skills to be able set up required model details. Although the learning ability can be helpful, it can never provide a full model transformation into source code. Also the transformation between different languages can lead to implementation errors during mapping and therefore it request higher testing demands and resources.

5.2. DSM result

DSM is based on narrow domain focus. The language of model comes directly from REA ontology. That allows performing validation and verification check and the model eliminates the need for mapping between any languages by using domain terms. Models are well readable for people working with REA ontology and they do not need any programming skill to create an executable application from model.

The biggest problem of this approach is limited number of semantic abstractions that the generator can contain. REA ontology contains a lot of different semantic abstractions and some of them replace some other, some of them cannot be used together etc. Therefore it is not possible to implement all combinations of them into generator. Another problem is that REA allows creating new semantic abstractions and every modification of domain language requires modification of generator and its recompilation and redistribution. Generator recompilation can cause older model incompatibility.

5.3. Comparing results

Development procedure of ontology systems using these technologies is different and each of

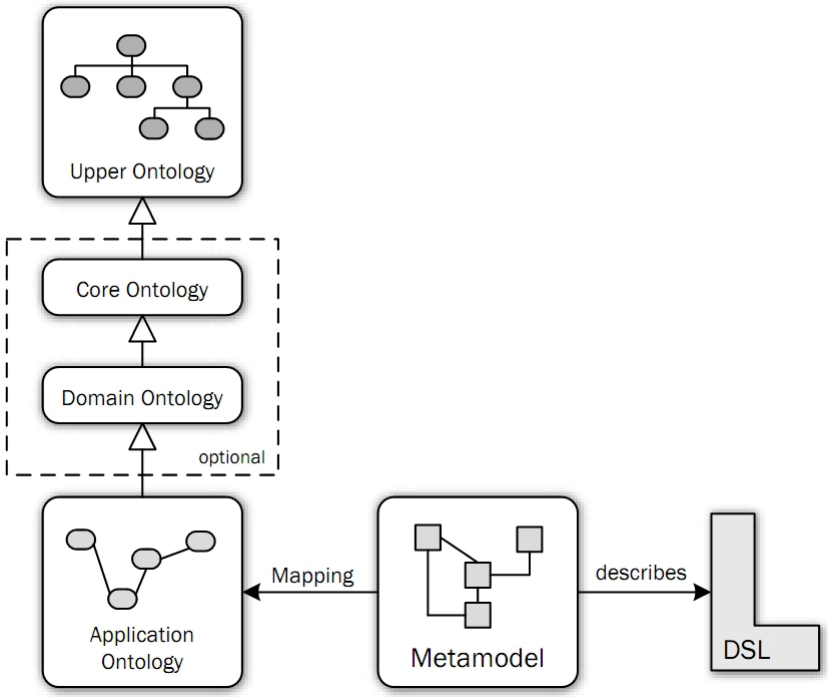

Figure 5. Ontological hiearchy [10]

them has positives and negatives. To model the general ontology, which assumes with possible changes in the structure, it is better to use the MDA technology but for the cost of no user friendly development and higher testing and resources requirements. In case of MDA each transformation between two languages has quadratic complexity $O(n^2)$ whereas the DSM model transformation has linear complexity $O(n)$ [10].

If only few semantic abstractions will be used (for example only one type of model will be created by application) it is much better to use DSM that has many advantages for REA ontology development. Nowadays there are some DSM modifications for supporting language changes without generator recompilation such as Expandable DSM generator [11].

The cost of development by MDA technology is equalized over whole developmental period. The DSM has largest costs at beginning of development, when the generator is built and subsequent development of applications based on the REA ontology has minimal costs because of higher abstraction level and thus increased productivity.

5.4. Extendable DSM generator

The basic idea of this approach is using some kind of pug-in system for extending the functionality of the generator. The modeling tool contains only basic semantic abstractions and all additional semantic abstractions are added through plug-ins (containing script for generator), which can be distribute via web services so modeling tool can automatically download and install needed packages. The solution of extendable DSM generator is described in [11].

6. Conclusion

Comparison of MDA and DSM technologies showed the fundamental differences and ways to using. Generally we can say that the REA ontology is better supported by DSM technology because it offers a lot of benefits such as using of the domain knowledge, the model validation and verification, low testing requirements and more. Due to architectural structure the DSM usage is limited by the narrow specific domain with the constant ontology and the limited count of

semantic abstractions. For that reason it is better to use the MDA technology if frequent changes of semantic abstractions of the REA ontology are expected. If large amount of modifications is not expected, but the ontology is not constant, the DSM extensible generator allowing changes of some semantic abstractions can be used [11].

Before the development begins it is necessary to perform basic analysis how will be the final system used and then choose which technology is better to that particular case.

Acknowledgement

The paper is supported by the grant reference no. 6141 provide by IGA Faculty of Science University of Ostrava.

References

[1] C. Chang and L. Ingraham, *Modeling and designing accounting systems: using Access to build a database*. John Wiley & Sons, Inc., 2007. [Online]. http://books.google.cz/books?id=NfzAAAAMAAJ

[2] T. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, Vol. Volume 5, No. 2, 1993, pp. 199–220, iSSN 1042-8143.

[3] D. Gasevic, D. D., and D. V., *Model Driven Architecture and Ontology Development*. New York: Springer Berlin Heidelberg, 2006.

[4] B. Bennett and C. Fellbaum, *Formal ontology in Information Systems*. IOS Press, 2006.

[5] P. Hruby, *Model-driven design using business patterns*, ser. Springer eBooks collection: Computer science. New York: Springer-Verlag, 2006.

[6] P. Hruby, M. Hucka, F. Huňka, J. Kasik, and D. Vymetal, *Víceúrovňové modelování podnikových procesů (Systém REA)*, ser. Series on Advanced Economic Issues. Ostrava: VŠB-TU Ostrava, 2010.

[7] H. Sedlackova and K. Buchta, *Strategicka analyza*. C H Beck, 2006.

[8] M. Fowler, *Domain-specific Languages*. Addison Wesley Longman, Inc., 2010.

[9] G. Guizzardi, L. Ferreira Pires, and M. van Sinderen, "On the role of domain ontologies in the design of domain-specific visual modeling langages," in *Proceedings of the 2nd OOPSLA Workshop on Domain-Specific Modeling Languages*, J. Tolvanen, Ed. Birmingham, USA: University of Alabama at Birmingham, 2002, pp. 25–38. [Online]. http://doc.utwente.nl/66750/

[10] M. Brauer and H. Lochmann, "Towards semantic integration of multiple domain–specific languages using ontological foundations," *Lecture Notes in Computer Science*, Vol. Volume 5021/2008, No. 2, 2008, pp. 34–48, available at WWW: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.91.9973.

[11] J. Žáček, Z. Meliš, and F. Huňka, "Extendable domain specific modelling generator based on web services," *ECON*, Vol. Volume 19, 2011, pp. 98–104.

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 15–24, DOI 10.5277/e-Inf130102

Efficient Adoption and Assessment of Multiple Process Improvement Reference Models

Simona Jeners*, Horst Lichter*, Carlos Gomez Rosenkranz*
*Research Group Software Construction, RWTH Aachen University
simona.jeners@swc.rwth-aachen.de, lichter@swc.rwth-aachen.de,
rosenkranz@swc.rwth-aachen.de

Abstract

A variety of reference models such as CMMI, COBIT or ITIL support IT organizations to improve their processes. These process improvement reference models (IRMs) cover different domains such as IT development, IT Services or IT Governance but also share some similarities. As there are organizations that address multiple domains and need to coordinate their processes in their improvement we present MoSaIC, an approach to support organizations to efficiently adopt and conform to multiple IRMs. Our solution realizes a semantic integration of IRMs based on common meta-models. The resulting IRM integration model enables organizations to efficiently implement and asses multiple IRMs and to benefit from synergy effects.

1. Introduction

Nowadays, the software market is expanding and clients are requesting better, faster, and cheaper software products. However, the Standish Group regularly reports that the failure rate of IT-projects is still too high: 68% of IT-projects do not meet their deadlines nor achieve the requested quality or are cancelled [1]. One important impact factor to project success is the quality of the applied IT-processes because software quality heavily depends on these processes. Hence, more and more organizations are obligated to identify, structure, and improve their processes systematically. Because the process improvement road is quite long and expensive it needs to be guided. To support process improvement different IT reference models such as CMMI [2], ISO/IEC 15504 [3], COBIT [4] or Functional Safety [5] may be considered and applied. Improvement Reference models (IRMs) are collections of best practices based on the experience and knowledge of many organizations. We call these best practices procedures. The IRMs are published as maturity-, procedure- or quality-models as well as standards or norms. Although IRMs exist for different IT areas, such as Software and System Development, IT Governance, Software Safety or IT Services, they may address similar topics. For example, project or risk management is addressed in almost all IRMs. The adoption of multiple IRMs allows an organization to exploit synergy effects between them. On the one hand organizations can address coordinately different and common areas. On the other hand the weaknesses of a single IRM can be overcome by the strengths of others.

Although there is free information available about each single IRM, there is no integrated solution that makes a collection of IRMs more transparent and supports organizations in the adoption and assessment of IRMs. This lack of transparency makes the effort for process management and assessment of multiple (evolving) IRMs unnecessary high. The main problems that hamper organizations to use the experience and knowledge reflected by IRMs are:

- **IRMs exist in different shapes, for multiple domains and cover many areas**. As already mentioned there are many types of IRMs, such as norms, quality-models or standards, which vary in certain areas but also share some ones in multiple IT domains. Organizations usually do not understand the similarities and differences between the IRMs in this collection.
- **IRMs are based on different structure and terminology**. Because IRMs are developed for different IT domains and by different institutions, each single IRM defines its own specific structure and uses a specific set of terms. Hence, different terms are used for the same semantic concept. Both, the different structure and different terminology hamper to understand and adopt IRMs.
- **IRMs may address similar topics**. Although IRMs exist for different IT areas, they may address similar topics. For example, project or risk management is addressed in almost all IRMs. To efficiently adopt multiple IRMs the organization must be able to easily compare the selected IRMs and identify their similarities and differences. Procedures of IRMs can be described either very generally or more concretely. The organizations should recognize similar procedures to better understand the abstract requirements of the more general one. Furthermore, organizations should be aware of the essence of similar procedures for a better overview and understanding. The specific details of each IRM should also be easy to identify if necessary.
- **IRMs are changing**. Since IRMs are updated continuously and new IRMs are developed organizations must keep pace with their evolution and must be able to understand and apply the changes.

1.1. Goals and Solution Approach

In order to solve the problems mentioned above we propose a new approach called MoSaIC (Model based Selection of Applied Improvement Concepts) aiming to achieve transparency and to support organizations in an effective adoption and assessment of multiple IRMs. The goals associated with MoSaIC are:

- **G1**: Facilitate the understanding and avoid misinterpretations of IRMs.
- **G2**: Identify similar procedures and extract their essence as abstract practices.
- **G3**: Provide traceability between abstract practices and their instances in the IRMs.
- **G4**: Allow an easy identification of the dependencies between procedures or process areas of different IRMs.
- **G5**: Support different levels of abstraction of IRMs.
- **G6**: Support an easy update of changed IRMs and an easy integration of new ones.

For short, we achieve transparency by a seamless and semantic integration of different IRMs. Although the integration of IRMs is a central issue of MoSaIC, it addresses further challenges as well, e.g. the systematic selection of IRMs or parts of IRMs that are best suited for an organization. To address this problem, not only IRMs but also other information (e.g. business goals or constraints) is integrated into MoSaIC. We focus here on MoSaIC's model based integration approach of IRMs only.

1.2. Related Work

The need of a process architecture in a multi-model context is mentioned in a series of articles from SEI [6]. This raises the awareness to define a generic and integrated model which allows describing different IRMs as well as organizational processes. This model should make IRMs more transparent and support organizations to find similarities between different IRMs. Basic elements mentioned in [7], [6] or [8] such as inputs, outputs, roles, their relations are part of our integration model as well.

Ferreira et al. [9] present an approach to achieve transparency of IRMs by comparing IRMs. The problems mentioned above, the different abstraction levels of IRMs, their overlapping, and their complexity are also mentioned. This approach tries to solve these problems by defining metrics to manually compare IRMs. A manually comparison of the IRMs

is performed in [10] to identify similar practices of different IRMs and their essence. We also provide support for identifying abstract practices but based on an automatic comparison. Our best practices can be easily traced to their instances in the different IRMs. The traceability is also addressed in a case study for the integration of large aerospace IRMs [11]. Here, different types of traceability between activity, input or output elements are defined. However, our approach extends one traceability type by considering the semantic relations between these elements. There are many contributions in the literature to the integration of IRMs and their comparison. For example in [12], [13] or [7], the authors define a common structure to link IRMs and reveal their similarities. For this purpose similar procedures of IRMs are connected manually. In contrast to the first two approaches we model on a more fine grained level and identify procedure' elements to support an automatic comparison. The third approach also addresses this fine granularity, it connects similar elements of different procedures but does not define what does similarity means. We differentiate between different similarityrelations to get a more accurate degree of similarity between IRMs. Soto and Münch [14] formalize and automatically compare the IRMs and internal processes but also by only considering the semantic equivalence between the basic elements (activities, stakeholder, and products). Their approach also addresses the problem of IRMs' evolution. Their comparison approach is used to support the changes of the internal processes or IRMs and to assure a continuously compliance.

The remaining of this paper is organized as follows. In the second section we describe the elements and relations of the MoSaIC's IRM integration approach. In section 3 we present excerpts of a MoSaIC case study applied to model parts of CMMI, COBIT and Functional Safety. Based on this case study we finally discuss the results of our evaluation and give an overview to future work. Conclusions and a summary conclude this paper in the last section.

2. The MoSaIC IRM Integration Approach

In the following we describe the MoSaIC way to integrate IRMs. First, we motivate and give a short overview of our integration approach. Then we present in detail the two meta-models of MoSaIC that provide the basis for a model based IRM integration approach.

The main idea of MoSaIC's IRM integration approach is to normalize IRMs based on a joint structure and on a common set of terms. According to mega modeling theory [15], we can normalize by defining appropriate meta-models. We have analyzed published IRM meta-models, e.g. the one of CMMI, extracted and added only elements that are sufficient to achieve the goals defined at the beginning of this paper.

To model different IRMs the same way we have developed the so called *Integration Structure Meta-Model* (IS Meta-Model). It defines core and additional IRM element types introduced by different IRMs as well as fine grained IRM concept element types, such as activities, artifacts or roles. While the core and additional element types allow providing a rough overview of the most important aspects of IRMs, the conceptual elements types allow the integration of concrete and abstract IRMs and a detailed comparison of IRMs. A *IRM concept* (concept for short) is a word or the smallest combination of words that has a unique meaning in the context of IRMs. For example "project plan" or "work breakdown structure" are concepts used in IRMs. Concepts can be derived from activities, roles, inputs and outputs of IRMs.

For each IRM, such as CMMI, SPICE, COBIT or ITIL, we have extracted the core, additional and conceptual information and created respective *IRM Integration Structure Models* (IRM-ISMs). Mappings from the single IRM-ISMs to the IRMs' original structures provide more information if needed.

Furthermore, IRMs should be modeled using the same terminology. We introduce a mechanism to translate and map the terms/concepts used by each single IRM to a common normative set of terms/concepts. For this purpose we

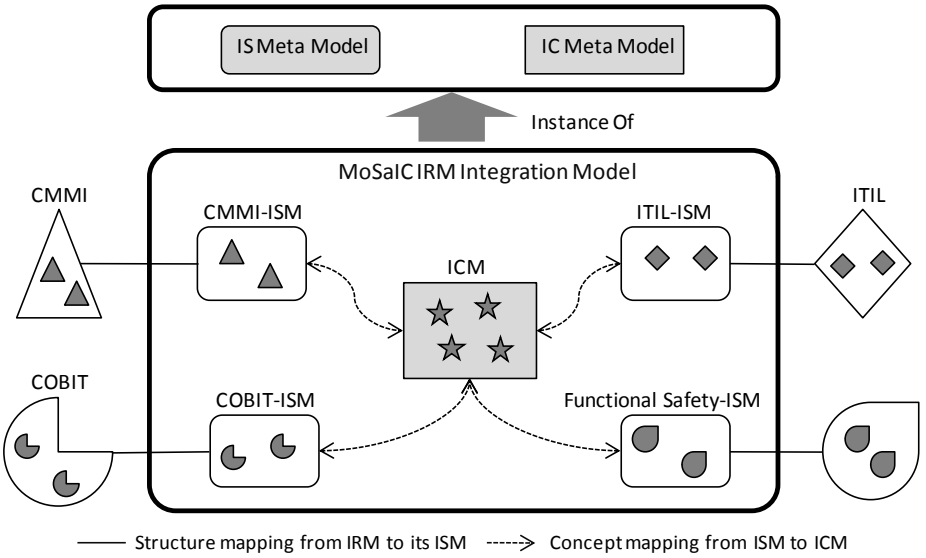

Figure 1. Architecture of the MoSaIC IRM integration approach

have created a model containing the closure of all IRM concepts, called *Integration Concept Model* (ICM). For each specific conceptual element in a IRM-ISM there is a corresponding general ICM concept. A general concept defined in the ICM can be mapped to several finer conceptual elements of one single IRM or of several different IRMs if these together are semantically equivalent to the general concept. The ICM concepts can be seen as dictionary entries having synonyms and explanations in the different IRM-ISMs. This conforms to what is called *Linguistic concordance* (list of words with their immediate contexts) and supports a better understanding and avoidance of misinterpretations of the IRMs' content. Obviously, the ICM is the sole instance of its meta-model, the *Integration Concept Meta-Model* (IC Meta-Model) and links all IRM-ISMs.

In order to model the semantic of IRMs appropriately and to further improve their comprehension we have enhanced our meta-models by attributes and semantic relations (e.g. to model similarity between concepts). For example, an ICM concept has a *definition* attribute (usually given by an expert) or ICM concepts may be related by a *generalizationOf* relation. To summarize, the ICM specifies the common concept language for IRMs and facilitates the understanding of their content.

Figure 1 schematically depicts the purpose and application of both meta-models and their respective concrete models IRM-ISMs and ICM. The different structures of IRMs are represented by different geometrical shapes while the different used terminology is symbolized by different small geometrical internal shapes. For each IRM a corresponding IRM-ISM is shown (e.g. CMMI-ISM) being part of the overall MoSaIC IRM Integration Model. All ISMs are instances of the IS Meta-Model. Hence, all ISMs use the same set of element types which makes them analyzable and comparable. ICM (the only instance of its IC Meta-Model) is part of MoSaIC's IRM Integration Model as well. It defines all concepts and semantically links all IRM-ISMs by connecting related concepts across the borders of single IRMs.

Figure 2 shows the most important elements of the ***Integration Structure Meta-Model***. For the representation we use a notation similar to UML class diagrams. We have grouped the elements in three packages:

1. **Core** contains elements mostly defined by meta-models of existing IRMs.
2. **Add-Ons** offers elements that are not always present in all IRMs.
3. **Concepts** contains elements to model concept information of IRMs on a fine grained level.

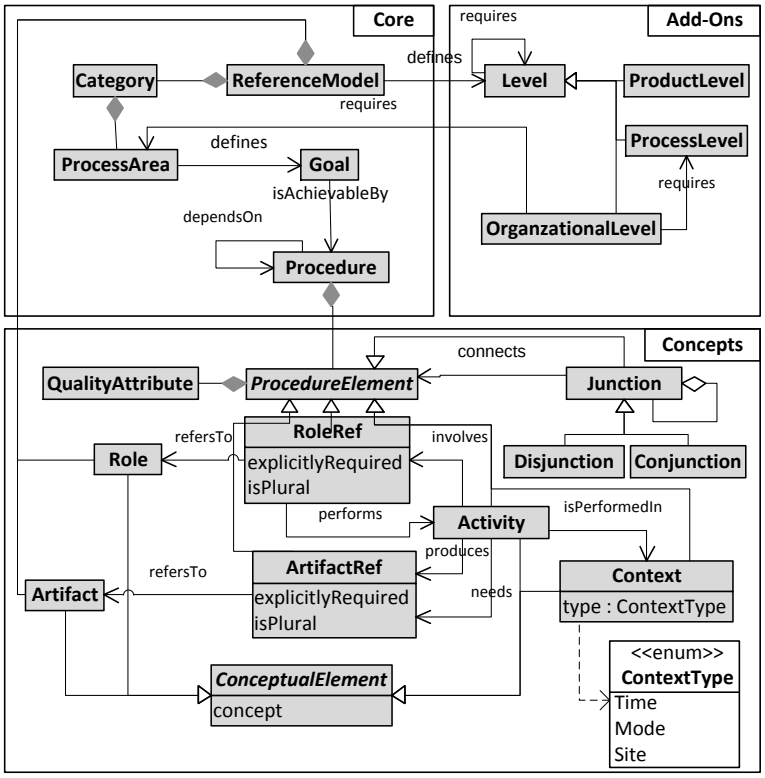

Figure 2. The Integration Structure Meta-Model

Package **Core**. The top element of this package is called *ReferenceModel*. It represents a certain IRM and is structured by means of *Categories*. A *Category* defines a certain topic that is addressed in one or more processes defined by the IRM. A *ProcessArea* addresses a topic to be improved and is part of exactly one *Category*. Each *ProcessArea* defines one or more *Goals*. The requirements to achieve the goals are described by *Procedures*. A *Procedure* defines one or more *Activities* with their *Roles*, *Inputs* and *Outputs* (called *Artifacts*). By means of the *dependsOn* relation dependencies between Procedures are modeled (e.g. if a procedure needs as input the output of another procedure).

Package **Add-Ons**. *ReferenceModels* may define Levels. A *Level* represents a degree that an organization can reach by applying the IRM. By means of the relation requires a hierarchy of levels can be modeled. Three special kinds of levels are defined: *OrganizationalLevel*, *ProcessLevel* (i.e. a level of a *ProcessArea*) and *ProductLevel*. An *OrganizationLevel* may require a certain *ProcessLevel* and may also require that certain *ProcessAreas* are established in the organization.

Package **Concepts**. Our approach to integrate different IRMs is centrally based on the notion of *Concepts*. Therefore, we model the specific conceptual elements of each single IRM in the respective ISM as well as their corresponding general concepts in the ICM. This enables to link similar specific concepts of different IRMs and to compare IRMs.

Activities, *Roles*, and *Artifacts* of IRMs are concrete *ConceptualElements*. An *Activity* may involve *Roles* and is performed by one or more *Roles*; it usually needs and produces *Artifacts*. Because a certain *Role* or *Artifact* can be used in different *Procedures* of a IRM, only their references are associated with *Activities*. Hence, *Activities*, *RoleRefs* and *ArtifactRefs* are the central aspects of a *Procedure*, abstractly modeled by class *ProcedureElement*. *ProcedureElements* have additional information that specifies their usage in *Procedures*. For example, they may be characterized by *QualityAttributes* (e.g. “formally approve the project plan”). Furthermore, they may

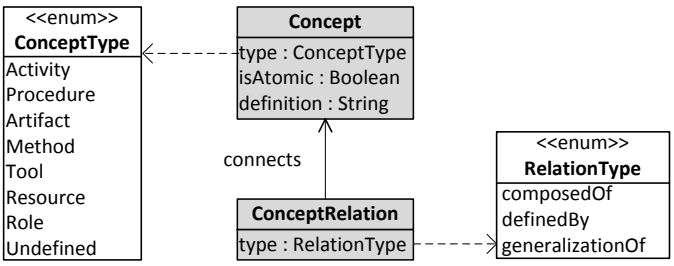

Figure 3. The Integration Concept Meta-Model

be connected by the logical relations *Conjunction* or *Disjunction*, because a procedure may require multiple *ProcedureElements* of the same type (e.g. "eliminate or minimize project risks"). By means of the *Junction's* self composition relation each combination of logical relations can be modeled (e.g. "carrying out the applicable overall, the E/EPS and software lifecycle phases").

However, there are some differences between the concrete *ProcedureElements*. One the one hand, *Roles* and *Artifacts* mentioned in a procedure can be *explicitlyRequired* or not. For example, in "define the project plan" the artifact "project plan" is explicitly required whereas "eliminate the faults in the software" may lead to model the artifact "software without faults" which is not explicitly required. We also model the multiplicity of an artifact or role (*isPlural*) to allow a precise comparison. On the other hand, *Activities* may be performedIn different *Contexts* (e.g. "approve the plan before project initiation"). As context information is conceptual information as well, *Contexts* are special *ConceptualElements* and as they are elements of procedures they are also special *ProcedureElements*. A *Context* usually explains its *Activity* (e.g. "maintain the programme by controlling the projects"), but it may also specify a temporal relation (e.g. "approve the plan before project initiation") or specify a local relation (e.g. "review requirements specification in the IT department"). The different context types are modeled by an attribute of type *ContextType*.

Figure 3 depicts the elements of MoSaIC's ***Integration Concept Meta-Model***. Although it has a pretty simple structure it is sufficient to model the world of IRM concepts with their relations. Obviously, a *Concept* (which is a term or a combination of terms from a IRM) is the main element. A *Concept* always has a *ConceptType* and may be related to other *Concepts* by so called *ConceptRelations* which are typed as well. The *ConceptType* determines the role of a *Concept* in a certain context (e.g. "work breakdown structure" may be an *Artifact* but also a *Method* depending on the context). *ConceptRelations* are used to model similarities between concepts. A *Concept* may be *composedOf* other *Concepts*. For example "requirements" is *composedOf* "functional requirements" and "non-functional requirements". Furthermore, a *Concept* may be a *generalizationOf* a more concrete *Concept* (e.g. the concept "stakeholder" is more general than "project manager"). In addition, a *Concept* may be *definedBy* other *Concepts*. For example, the concept "plan the involvement of stakeholder" is defined based on the concept "stakeholder". If a *Concept* is self-contained, it is called atomic (attribute *isAtomic*). Atomic concepts are usually defined by experts. Currently, initial sets of concept and concept relation types are offered. This architecture is flexible and open to introduce new concept and concept relation types if needed.

In the next section we describe the application of the both meta-models to ease their understanding and to show their modeling abilities.

3. Application of the Meta-Models

In order to evaluate the appropriateness of the developed meta-models we have performed a mid-size case study considering some parts of COBIT, CMMI and FS.

Although each IRM focuses on a specific IT domain they are similar in certain aspects. Be-

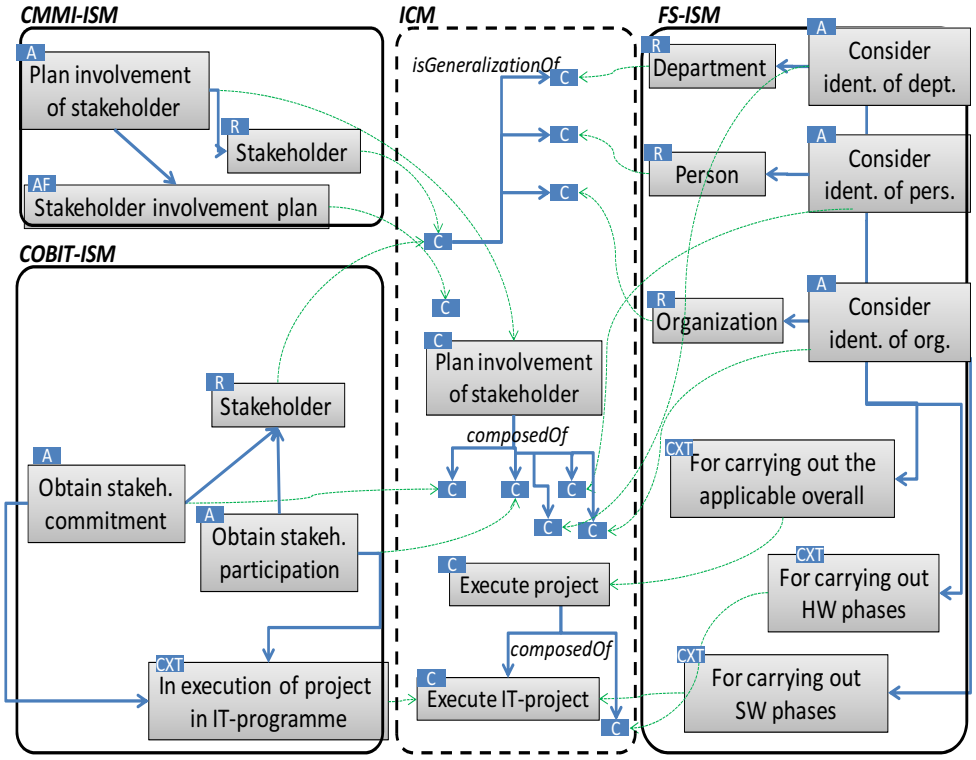

Figure 4. Excerpt of a MoSaIC IRM Integration Model

cause the aim of our case study was to explicitly show the similarities between the considered IRMs and therewith their integration, we selected designated procedures that contain similar conceptual elements: we consider procedures the following IRMs-elements: CMMI practices, the sentences of COBIT control objectives and Functional Safety requirements (CMMI PP SP2.6: "Plan the involvement of identified stakeholder" with the typical work product "stakeholder involvement plan"; COBIT PO10.4: "Obtain commitment and participation from the affected stakeholders in the (..) execution of the project within the context of the overall IT-enabled investment programme"; Functional Safety, Part 1, 6.2.1b: "Consider the identification of the persons, departments and organizations which are responsible for carrying out (..) the applicable overall, E/E/PES or software safety lifecycle phases").

Figure 4 depicts the developed models. Each ISM contains conceptual elements, such as activities (A), artifacts (AF), or roles (R). The COBIT- and FS-activities are described by additional context information, contexts (CXT) are added and linked to the respective activities. The associations of ISM conceptual elements with their correspondent concepts (C) in the ICM integrate the procedures of the different IRMs. For example, the CMMI-ISM activity "Plan involvement of stakeholder" is composed of the COBIT- and FS- activities. Another example of similarity between the modeled procedures is given by the role "Stakeholder". While CMMI and COBIT use this term, it is represented in FS by the three roles "Departments", "Persons" and "Organizations". Therefore, we can easily identify the similarities between these procedures. Furthermore, we can easily extract the essence by identifying only the general concepts: perform the activity "plan the involvement of the stakeholder" for the "execution of the project", involve the "stakeholder" and produce the output "stakeholder involvement plan". The ICM allows identifying the relations of these general concepts to the more concrete concepts in the ISM of the corresponding IRMs: the organization can easily identify the details (e.g. in FS the stakeholder are represented by persons, departments, organizations). Therefore, the traceability between the abstract practice

and the more detailed instances in the IRMs is supported.

The dependencies between the procedures of a IRM or different IRMs can also be identified. The procedures that share similar artifacts are interdependent: one procedure uses the output of another procedure as input. For example the CMMI procedure PP 2.7 “Establish and maintain the overall project plan” produces the output “project plan” that is used as input in the COBIT procedure PO10.7 ”The project plan (...) should be approved in line with the programme and project governance framework”.

4. Experiences and Future Work

In the following we describe the experiences gained with the developed IRM integration approach. Furthermore, we propose some ideas towards further research and ideas concerning the application of the MoSaIC IRM integration approach.

4.1. First Experiences

The modeling of selected parts of CMMI, COBIT and Functional Safety showed that the IS and IC meta-models offer a stable structure for integrating the chosen IRMs. However, modifications of the meta-models may be possible after gaining more experience in modeling further IRMs.

The IRM specific ISMs as well as the central ICM were created manually. Because our approach is based on fine grained IRM information, a large number of conceptual elements had to be modeled. Thereby, redundant definitions of semantically equivalent concepts in the ICM had to be avoided. Furthermore, we always tried to relate new concepts to existing ones in case of similarity (by “composedBy”, “generalizationOf” or “definedBy” relations). However, the manual modeling was not always easy and sometimes it was difficult to model concepts consistently.

Based on our experience we evaluate our IRM integration approach in relation to the goals listed in section 1.1 as follows.

The created ISMs and the ICM allow a semantic integration of IRMs. Because each IRM-ISM contains the most important IRM information it provides a condensed overview for organizations (G1). Furthermore, the ICM eases the understanding and avoids misinterpretations of terms and concepts used in IRMs, because the concepts are associated with their synonyms and contexts in the IRM-ISMs. Every new IRM may enrich the description of a concept and ease its comprehension. Therefore, the more IRMs are integrated, the smarter the ICM will become. This approach is a kind of crowdsourcing where organizations work together to improve the ICM by model-ing more and more IRMs. Furthermore, the created ICM connects the different terms used in the IRMs and connects the procedures of different IRMs. Therefore, the organizations can easily identify the dependencies between these different IRMs (G4).

Our approach supports a detailed comparison of IRM procedures. The fine grained model elements, the concepts connecting the procedures, and the semantic concept relations allow identifying similar procedures and their essence (G2). The extracted abstract practices help organizations to avoid redundancies and to reduce the effort in the adoption and assessment of multiple IRMs. The implementation of the abstract practices of certain IRMs assures the adoption of these IRMs. If necessary, the organizations can easily identify the details in the IRMs with the help of the traces (G3). These traces are useful also in the assessments. By considering the abstract practices of certain IRMs, it can be roughly assessed if these are implemented in the organizations. Finally, the details of the IRMs can be traced and be assessed.

The fine granularity of the models (ISMs and ICM) enables to model IRMs on different levels of abstraction: abstract, concrete IRMs and even internal processes can be easily integrated in the MoSaIC Integration Model (G5). This integration supports a better understanding of a certain area that is addressed by both of them (G1). Finally, changes on existing IRMs or new IRMs can be integrated in the MoSaIC Integration Model (G6). This is done by creating or updat-

ing the respective ISM and adding or updating the connections between the changed or newly created ISM and the central mediator, the ICM. The integration of a new IRM implies adding new concepts to the ICM in case they are not already defined.

4.2. Future Work

The proposed approach realizes a sound basis to integrate IRMs. However, more research has to be done to fully achieve all defined goals and to facilitate the application of MoSaIC's IRM integration approach.

Because the manual modeling of IRMs is a time consuming and error-prone process, a dedicated tool box is needed. Currently we have developed only sparse tool support based on the implementation of the meta-models as Ecore models in the Eclipse Modeling Framework [16]. By means of the respective generated tree-based editors we are able to manually create and maintain the ISMs and the ICM. A more sophisticated tool box should support the IRM expert e.g. to cope with the consistency problem and to model the fine grained conceptual elements. Furthermore, a semiau-tomatic tool performing a syntactical and linguistic analysis of the IRM documents may generate recommendations to model the conceptual elements properly. For example, prepositions like "based on" or "in line with" may require modeling a respective artifact. Because the IRM documents are written very differently, modeling recommendations can not only rely on syntactic rules. For example, in some IRM documents the activities of procedures are written by nouns while in others verbs are used. Hence, a plain syntactical analysis of the documents is not sufficient. Further rules are needed to transform the language used in the original documents in a "normalized" language. We will investigate if rules used to precisely write requirements ([17]) could be adopted to transform the original text in a "normalized" language (e.g. the passive or noun form of a verb is transformed in its active form). This may allow a semiautomatic extraction of conceptual elements and their modeling in the Integration Structure and Concept Model. Furthermore, the tool box should be able to adapt and improve its generated recommendations according to modeling decisions done by IRM experts.

Currently we are developing a new approach to compare IRMs based on the information stored in our models. At the moment, the identification of similarities and the comparison has to be done manually be an expert. Our new approach should enable a tool supported automatic comparison of IRMs and the determination of coverage degrees of procedures, respectively of considered IRMs.

5. Conclusions

In this paper we have presented MoSaIC, a model based approach to integrate reference models. The core idea is to represent each IRM in a dedicated Integration Structure Model (ISM) and the common concepts in one central Integration Concept Model (ICM).

The IRMs' integration achieves transparency and reduces complexity of IRMs. As the IRM-ISMs are instances of their common meta-model they normalize the different structures of the IRMs and provide a rough overview. Due to the modeled fine granularity, the common structure enables to semantically integrate different IRMs. The ICM normalizes the different concept terminology of the IRMs and forms a landscape of IT terms and their relations. Through common concepts and their semantic relations, this model facilitates the understanding of IRMs. A better understanding is supported also by the integration of general and concrete IRMs. Furthermore, it supports a detailed comparison of IRMs that allows to identify similar and interdependent procedures and helps organizations to avoid redundancies in the adoption and assessment of multiple IRM. Therefore, they will be able to efficiently and asses adopt multiple IRMs.

First experience and results with the presented approach are promising; we were able to effectively model the integration of some process areas of three IRMs. We expect that the results of our future work will make the integration of IRMs more accurate and comfortable.

References

[1] "CHAOS summary 2009," The Standish Group, Boston, Tech. Rep., 2009. [Online]. http://www.statelibrary.state.pa.us/portal/server.pt/document/690719/chaos_summary_2009_pdf

[2] "CMMI for development, version 1.3," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI 2010-TR-033, ESC-TR-2010-033, 2010. [Online]. http://www.sei.cmu.edu/reports/10tr033.pdf

[3] *ISO/IEC 15504 Information Technology Process assessment. Part 1: Concepts and vocabulary, Part 2: Performing an assessment, Part 3: Guidance on performing an assessment, Part 4: Guidance on use for process improvement and process capability determination, Part 5: An exemplar Process Assessment Model*, ISO/IEC Std., 2007.

[4] "Control objectives for information and related technology version 4.1," ISACA, Tech. Rep., 2007. [Online]. https://www.isaca.org/bookstore/Pages/Product-Detail.aspx?Product_code=CB4.1

[5] *IEC 61608 or Functional safety of electrical/electronic/programmable electronic safety-related systems*, IEC Std., Rev. 2, 30 April 2010.

[6] J. Siviy, P. Kirwan, L. Marino, and J. Morley, "Process architecture in a multimodel environment. white paper," Software Engineering Institute, Tech. Rep., 2008. [Online]. http://www.sei.cmu.edu/library/assets/multimodelSeries_wp4_processArch_052008_v1.pdf

[7] D. Malzahn, "Assessing – learning – improving, an integrated approach for self assessment and process improvement systems," in *ICONS 09, the Fourth International Conference on Systems*. Gosier, Guadeloupe, France: IEEE Computer Society Conference Publishing Services, 2009, pp. 126–130.

[8] Y. Wang, G. King, A. Dorling, and H. Wickberg, "A unified framework for the software engineering process system standards and models," in *4th IEEE International Software Engineering Standards Symposium and Forum*, Curitiba , Brazil, 1999.

[9] A. Ferreira, L. Machado, and M. C. Paulk, "Quantitative analysis of best procedures models in the software domain," in *17th Asia Pacific Software Engineering Conference*, 2010, pp. 433–442.

[10] Y. Wang, G. King, and H. Duncan, "Deriving personal software processes from current software engineering process models," in *European software process improvement: proceedings of the EuroSPI 2000 Conference*, Copenhagen, Denmark, 2000.

[11] O. Armbrust, A. Ocampo, J. Münch, M. Katahira, Y. Koishi, and Y. Miyamoto, "Establishing and maintaining traceability between large aerospace process standards," in *5th International ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, 18 May 2009, pp. 36–40.

[12] A. Ferchichi, M. Bigand, and H. Lefèbvre, "An ontology for quality standards integration in software collaborative projects," in *Model Driven Interoperability for Sustainable Information Systems*, J.-P. Bourey, Ed., Montpellier, FRANCE, 2008, pp. 17–30. [Online]. http://ceur-ws.org/Vol-340/paper02.pdf

[13] L. Liao, Y. Qu, and H. Leung, "A software process ontology and its application," in *International Workshop on Future Software Technology (IWFST-2005)*, Shanghai, 2005.

[14] M. Soto and J. Muench, "Using model comparison to maintain model to standard compliance," in *ICSE Workshop Comparison and Versioning of Software Models (CVSM 2008)*, Leipzig, Germany, 17 May 2008.

[15] J.-M. Favre, "Megamodelling and etymology," in *Transformation Techniques in Software Engineering*, ser. Dagstuhl Seminar Proceedings, J. R. Cordy, R. Lämmel, and A. Winter, Eds., No. 05161. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. [Online]. http://drops.dagstuhl.de/opus/volltexte/2006/427

[16] D. Steinberg, *EMF : Eclipse Modeling Framework*. Upper Saddle River, NJ: Addison-Wesley, 2009.

[17] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals A study Guide for the Certified Professional Requirements Engineering Exam Foundation Level IREB compliant*. Rocky Nook, Inc, Santa Barbara, 2011.

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 25–33, DOI 10.5277/e-Inf130103

A Knowledge-Based Perspective for Software Process Modeling

Noureddine Kerzazi*, Mathieu Lavallée*, Pierre-N. Robillard*

*Laboratoire de Recherche en Génie Logiciel, 2500 chemin de Polytechnique, Montr[Pleaseinsert\PrerenderUnicode{Âľ}intopreamble]al H3T 1J4, Canada, École Polytechnique de Montréal

Noureddine.Kerzazi@Polymtl.ca, Mathieu.Lavallee@Polymtl.ca, Pierre.Robillard@Polymtl.ca

Abstract

As the acquisition and sharing of knowledge form the backbone of the software development process, it is important to identify knowledge discrepancies between the process elements. Explicit representation of the knowledge components within a software process model can provide a means to expose these discrepancies. This paper presents an extension of the Software and System Process Engineering Metamodel (SPEM), to be used as a new knowledge modeling layer. The approach, which is based on ontologies for knowledge representation, constitutes an explicit method for representing knowledge within process models. A concept matching indicator shows the state of the process model in terms of the concept gaps for each task within the process. This indicator could lead to more informed decision making and better management of the associated risks, in terms of team competency, documentation quality, and the training required to mitigate them.

1. Introduction

Software engineering is knowledge-intensive [1], and so the acquisition and sharing of knowledge form the conceptual backbone of software process modeling. However, the latest version of SPEM [2], the Object Management Group (OMG)'s de facto standard devoted to software process modeling, does not support this knowledge concern. We tackle the issue in this paper by presenting an extended SPEM framework which focuses on the knowledge-oriented perspective of process modeling. Our approach responds to the need to provide process engineers with the means to perform knowledge assessments within processes, and we do this by integrating knowledge concepts within SPEM-based software process modeling. This knowledge-oriented perspective has been integrated into our previous work related to the implementation of a domain-specific language for software process modeling (DSL4SPM tool) [3].

In addition, the paper tackles the issue of knowledge discrepancies in the software process model, with the purpose of providing an indicator of a knowledge gap relating to any activity represented in that model.

The benefit of introducing knowledge concepts into software process modeling is to enable the identification of knowledge gaps between prescribed activities and the available resources. When such a gap is identified, the project manager can evaluate the risks it represents, and provide the developers with additional training as required [4]. The objective of this SPEM extension is thus to provide process engineers with a mean to document domain knowledge constraints directly into their process models.

The paper is organized as follows: Section 2, Related Works, presents theoretical background related to ontologies and knowledge representation. Section 3, Extension of SPEM, describes the SPEM extension developed to enable knowledge integration and gap measurement. Section

4 presents the visualization of the knowledge gap measure, along with its interpretations. Finally, section 5 presents concluding remarks and suggestions for future work.

2. Related Works

The background theory of this work relates to three main areas: software process modeling, knowledge management theories, and the use of ontologies. Knowledge management has been extensively studied, in the social sciences literature in particular. The purpose of this section is to highlight the salient concepts of the theories on which this approach is based.

2.1. Knowledge Management Theories

According to Davenport and Prusak [5], Knowledge Management (KM) can be defined as a process that relates to three issues: knowledge creation, knowledge representation, and knowledge sharing. This subsection highlights representative theories from the fields of management and the cognitive sciences, management theories emphasizing knowledge creation and representation, and cognitive theories emphasizing knowledge representation and storage.

In considering the management point of view, Nonaka & Takeuchi [6] identify two types of knowledge: tacit (T) and explicit (E). Tacit knowledge is personal and context-specific. It cannot, by definition, be explicitly expressed by an individual, but it frames his behavior. In contrast, explicit knowledge is externalized knowledge, and it supports communication, either formally or informally, and does so independently of who "knows". Within a software process, tacit knowledge resides in roles, while explicit knowledge resides in artifacts.

In considering the cognitive point of view, Novak and Canas [7] define knowledge as a structured set of interrelated concepts. They argue that learning involves the assimilation (i.e. internalization) of new concepts into existing cognitive structures. This theory is also embraced by cognitive psychologists, who assert that we learn by assimilating new concepts and propositions into existing cognitive structures, rather than through rote memorization [8]. So, representing knowledge in a structure of concepts [9] is in agreement with how cognitive psychologists believe we store knowledge in the brain.

Moreover, concepts can be organized in an ontology to simplify manipulation, sharing and reuse. An ontology is defined as "a formal explicit description of concepts in a domain" [10].

According to Anderson [11], there are two kinds of knowledge: declarative knowledge and procedural knowledge. Declarative knowledge is the factual or conceptual knowledge that a person has. It can be described and shared with others (e.g. facts about a programming language). Procedural knowledge is the knowledge of how to perform tasks, and focuses more on action than on information. This second type of knowledge is difficult to describe, but is nevertheless important, particularly in problem-solving (e.g. the experience of using a debugger). Robillard [1] presents an integrated view of various knowledge concepts in software engineering. Even though there is an abundance of theoretical knowledge models in the literature, there are few methods for integrating knowledge into the software process modeling field, as reported by Bjørnson & Dingsøyr in their systematic review [12]. With the recognition that knowledge is the primary source of an organization's innovation potential, our aim in this work is to superpose the knowledge-oriented perspective over the activity-oriented layer of the software process model, in order to provide a method for representing and managing knowledge in the software development process. The first step is to extend the SPEM-based processes with attributes related to knowledge. In this way, cognitive theory is used to represent:

- the knowledge required to carry out a task (considered as the core of the action),
- the knowledge provided by the artifacts and roles linked to this task.

The second step is to compare the knowledge required for the SPEM elements, such as artifacts, roles, and tasks, in order to identify gaps. The final step is to visualize the knowledge matching between tasks.

2.2. Ontology for Knowledge Representation

We chose an ontological representation among other formalisms for three reasons. First, cognitive psychologists, like Ausubel [8], affirm that people do not learn by rote memorization, but rather by summarizing, structuring, and relating concepts, and then assimilating them into existing cognitive structures. Based on this theory, the ontology domain was developed with the main aim of building knowledge repositories using a tree of interrelated concepts [13]. Second, we want to represent knowledge bases in a modular way, so that the model would be extendable and scalable to accommodate new items in the future. Third, we need a standard representation (e.g. RDF/XML or OWL 2.0) to be able to share knowledge bases and reuse them in different contexts.

Recently, more and more researchers have been using ontologies to understand software processes [14, 15]. According to Anquetil et al. [14], software system maintenance is a knowledge-intensive task, which can be supported with an ontology, and such an ontology can provide a good understanding of the application. The authors argue that many of the difficulties associated with software maintenance originate from knowledge management problems, and they propose a technique for knowledge extraction. Ferreira et al. [15] propose the integration of ontologies into the model driven architecture (MDA) paradigm.

In this work, we implement an interface, in addition to the basic process modeling one, to import multiple external ontologies.

3. Extension of SPEM

SPEM 2.0 is the OMG's standard for software process modeling [2]. It is based on UML and is dedicated to describing the components of software processes. According to its specification, a software process is a set of activities, each of which is composed of one or more tasks performed by an abstract active entity, called a Role. The Role is responsible for one or more tangible entities, called Work Products. Roles describe the responsibilities and a set of skills to facilitate their assignment to real people. Work Products are pieces of information produced by, or used by, Tasks.

However, SPEM supports activity-oriented modeling, which focuses on a structural breakdown of activities, and not knowledge modeling. There is therefore a need to extend SPEM to take into account a knowledge-oriented modeling perspective.

3.1. Integrating Knowledge Component into SPEM

Figure 1 shows the conceptual integration of the knowledge component into SPEM, the gray classes coming from the SPEM 2.0 specification. Note that *Work Product Use* is the generic term for the inputs and outputs of tasks, such as Artifacts, in the specification. The concept-based knowledge repository is structured as a tree of the *ConceptNode* element.

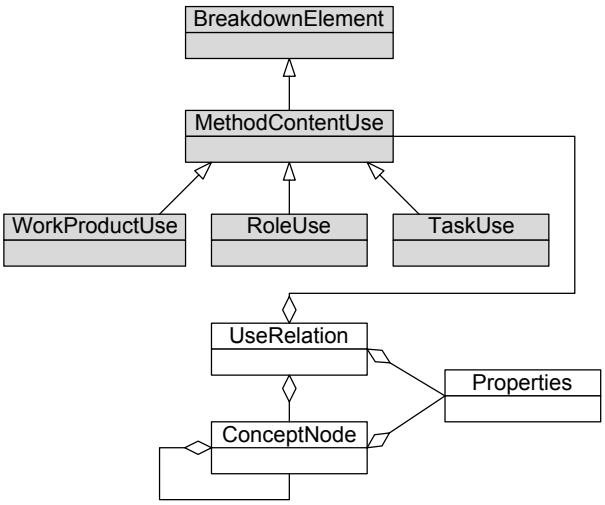

Figure 1. Extension of SPEM to support ontology. Elements in grey derive from the SPEM specifications. The extension is in white

The *ConceptNode* class represents a concept of the repository's tree-structured ontology. A node is associated with a list of properties. The *ConceptNode* is linked to process elements through the Relationship class. The Relationship class has one property, which enables the mod-

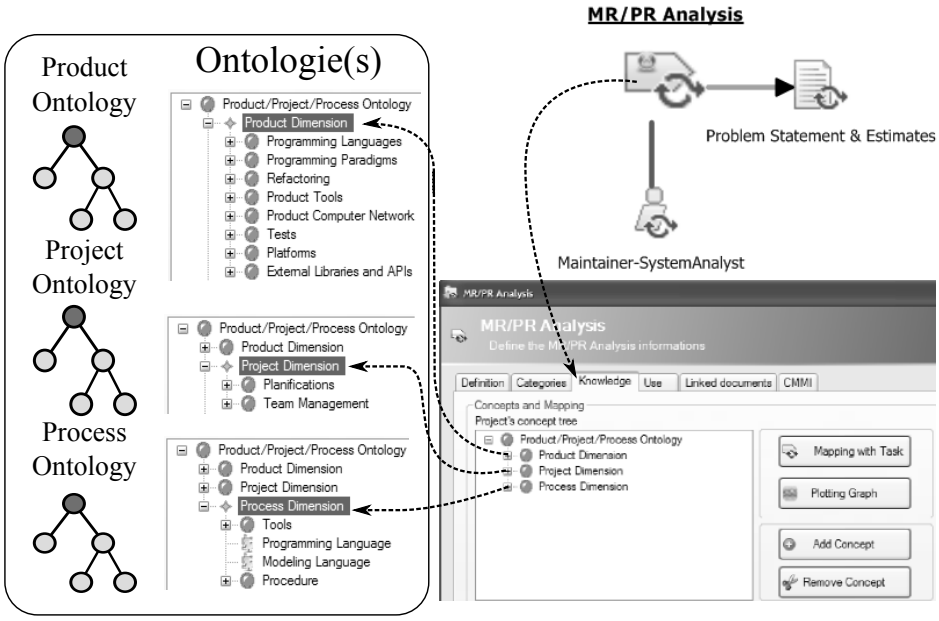

Figure 2. The specific form used to indicate the knowledge concepts needed for a task

eler to assign concepts to the process elements in a declarative way, a procedural way, or both. In this model, concepts do not yet have properties. The only relationship possible between concepts is an inheritance relationship,"*is-a*" . Future extensions may permit the expression of more properties and relationships.

According to Nosek and Roth [16], a visual representation of knowledge is clearly better for comprehension and conceptualization than a textual one. This paper presents visual modeling instead of the textual descriptions common in a number of specialized knowledge tools.

Figure 2 shows the specific form that allows knowledge modeling. The task is the core element of the process model; the process engineer has to indicate which knowledge concepts are needed to carry out each task, and, for each concept, the relationship with the process element (e.g. procedural and/or declarative) must be specified. The process engineer retrieves the related concepts required for each task from the concept ontology.

As shown in the upper-left corner of Figure 2, the ontology is organized according to three dimensions: product, project, and process. This multi-dimensional ontology approach (i.e. merging of ontologies) was motivated by validation experiments. The feedback of the participants revealed the need for a different level of knowledge. Some participants wanted a process ontology (e.g. role skills, artifacts, and guidance). Others asked for a project management ontology (e.g. assignment of human resources to roles, competency management). Still others were interested in a product ontology (e.g. API documentation, programming languages, techniques for understanding the source code, etc.). The results showed the need for a flexible and articulated tool, which can accommodate ontologies for multiple levels of knowledge.

3.2. Concept Maps for Knowledge Representation

The following four steps are required to evaluate the concept mapping between SPEM elements, which refers to process entities such as Activities, Roles, and Artifacts, as defined in the SPEM specification.

1. **Parameterization.** The first step is to specify the ontology, which is the formalization of the concept structure. The process engineer first loads a default or adapted concept tree. Then, the project team gathers and organizes the knowledge concepts relevant to the project context. As seen in Figure 2,

three concept trees are proposed by default, each one linked to an abstract level of knowledge (process, project, and product). Each concept may have a finer-grained structure represented with a number of attributes.

2. **Modeling.** For every task in the process model, references are set to a subset of concepts required for achieving the task. For every concept, the process engineer specifies the way in which the concept is required, following Anderson's model: declarative and/or procedural. For all incoming links to each task (e.g. from Work Products, Roles, Guidance), references are set to a subset of concepts provided by this element.
3. **Compute mismatch.** For every task, the system searches all incoming links, retrieves the concepts provided, and compiles the results. A concept required by the task is considered fully mapped if it can be provided by at least one of the input elements linked to this task. It is considered inadequately mapped if it is partially mapped (e.g. provided as declarative, when it is required to be procedural). Finally, it is considered not mapped if it is not provided by any SPEM element linked to this task.
4. **Visualize the results.** The concept matching indicator displays the results of concept mapping (as shown in Figure 3 below, which depicts knowledge mismatches). The user can visualize the resulting mismatches between prescribed knowledge and the available resources.

3.3. Algorithm for Knowledge Mapping

A formalism based on the conventional Vector Space Model (VSM) [17] is used to enable a simple visualization of the knowledge mismatch observed. VSM is used to compute the similarity between the knowledge required to carry out each of the prescribed tasks and the knowledge provided by the available roles, artifacts, and guidance. Cosine similarity measures are used to compute the similarity between two vectors of concepts characterized by n attributes (n-dimensional space). The results of this mapping are used to build the concept matching indicator. The first step is to link the process model to the ontology file, which contains a set of n concepts (Cn) relevant to the whole process. Thus, the knowledge required to carry out a given task ti is represented by a vector:

$$\vec{t_i} = \{C_1, C_2, ..., C_n\} * \vec{I} \tag{1}$$

where $I_i = 1$ if the concept is required, and 0 if not. According to Anderson [10], concepts can be classified in two categories: procedural or declarative, both of which are denoted by a vector (p,d). To simplify the illustration, we present only the Anderson attribute for the knowledge concept, which will be sufficient for most practical purposes. Let t'_T be the vector representing a set of typed concepts required for task T:

$$\vec{t'_T} = \{C_1(p,d)_{t'}, C_2(p,d)_{t'}, ..., C_n(p,d)_{t'}\} * \vec{I} \tag{2}$$

The next step is to link the source elements (i.e. other linked SPEM elements, such as Role, Artifact, and Guidance) of each task. These elements gather the concepts provided, instead of the concepts required. Let p'_E be the vector representing a set of typed concepts provided by a given element E:

$$\vec{p'_E} = \{C_1(p,d)_{p'}, C_2(p,d)_{p'}, ..., C_n(p,d)_{p'}\} * \vec{I} \tag{3}$$

The similarity between the vector t'_T and the p'_E of each of the elements provided is obtained by the cosine calculation. This gives a way of interpreting the quality of the mapping between the set of required knowledge concepts for each Task and the set of knowledge concepts provided.

$$\begin{aligned} \| p'_E \| &= \sqrt{C_1(p,d)^2_{p'} + ... + C_n(p,d)^2_{p'}} \\ \| t'_T \| &= \sqrt{C_1(p,d)^2_{t'} + ... + C_n(p,d)^2_{t'}} \\ D &= \sqrt{\sum_{i=1}^{n} (C_i(p,d)_{t'} - C_i(p,d)_{p'})^2} \\ \cos\theta &= \frac{\| p'_E \|^2 + \| t'_T \|^2 - D^2}{2 \| p'_E \| \| t'_T \|} \end{aligned} \tag{4}$$

where the cosine values vary between 0 and 1:

p' values	t' values	Deviation
C1(p)+C2(d)+C3(p,d)	C1(p)+C2(d)+C3(d)	30°
C1(p,d)+C2(d)+C3(d)	C1(p,d)+C2(d)+C3(d)	0°
C1(p,d)+C2(p)+C3(p)	C1(p)+C2(p,d)+C3(d)	60°

Table 1. Calculation example

- If the required and provided concept vectors coincide completely, which means a perfect, complete concept mapping, the cosine is equal to 0.
- If the two vectors have no concepts in common, it means that all the concepts are unmapped and/or unused, and the cosine is equal to 1.
- Otherwise, the result mapping should be between 0 and 1.

The cosine angle values range between 0° and 90°. To improve readability all angle values are doubled, which enables a two quadrants (180°) representation, as shown in Figure 3.

Table 1 provides some examples of the calculation method.

4. Concept Matching Visualization Examples

As illustrated in Figure 3, the graph is divided into three zones (dark grey, medium grey, and light grey). An arrow pointing toward the right and within the light grey zone indicates a good match. An arrow pointing toward the left and within the dark grey zone indicates a poor match. The size of these colored zones is customizable, however, and the modeler can indicate the acceptance threshold for each zone. Each arrow represents a task of the process. The angle of the arrow represents the degree of knowledge matching for the given task. For example, the *Test Plan Writing* task (a horizontal arrow pointing to the right) is completely mapped, which means that all the concepts required for this task are provided, while the *UI Specification* task (a horizontal arrow pointing to the left) presents a complete mismatch between the prescribed concepts required to perform this task and the concepts available within the team resources. There can be as many zones as needed by the project.

4.1. Acceptable Match

The *Behavioral Model Creation* task, shown in Figure 3, is in the acceptable zone. This task is concerned with the creation of use cases, sequence diagrams, state diagrams, etc. To perform this task, a *modeling language* concept (C1) is required for both declarative and procedural knowledge. Declarative knowledge is needed to understand the theory behind this modeling language and procedural knowledge will enable the role to perform the task. Similarly, an *analysis technique* concept (C2) is required. A *guideline* concept (C3) is also required, since the diagrams produced must follow the templates of the organization. Basic declarative knowledge on the *requirement elicitation technique* (C4) is also required since the input of this task is the requirements document. The t'_T vector of this task is therefore:

$$\vec{t'_T}=\vec{C_1}\ (p,d)_{t'}+\vec{C_2}\ (p,d)_{t'}+\vec{C_3}\ (p,d)_{t'}+\vec{C_4}\ (d)_{t'} \quad (5)$$

It is found that this is not fully consistent with the concepts provided to the task. The roles are proficient in terms of the *modeling language* (C1), as they both have declarative knowledge and procedural experience. This is, however, a transformation task, requiring the transformation from requirements into diagrams. The roles are not familiar with this challenge, but hands-on training with a professional is planned. Declarative and procedural support for the *analysis technique* (C2) is therefore ensured. The requirements document presents declarative facts on the *guideline* to be used (C3). Is is also well enough detailed as to ensure that the *requirements elicitation technique* (C4) is well defined. The p'_E vector of this task is therefore:

$$\vec{p'_E}=\vec{C_1}\ (p,d)_{p'}+\vec{C_2}\ (p,d)_{p'}+\vec{C_3}\ (d)_{p'}+\vec{C_4}\ (d)_{p'} \quad (6)$$

The angle is then measured between the two vectors. Formula 7 shows the vectors in coordinate form, grouped by concepts, as well as the cosine and angle calculations.

$$\begin{aligned}
\vec{t'_T} &= \{(1,1)_{t'}; (1,1)_{t'}; (1,1)_{t'}; (0,1)_{t'}\} \\
\vec{p'_E} &= \{(1,1)_{p'}; (1,1)_{p'}; (0,1)_{p'}; (0,1)_{p'}\} \\
\cos\theta &= \frac{\| p'_E \|^2 + \| t'_T \|^2 - D^2}{2 \| p'_E \| \| t'_T \|} \qquad (7) \\
&= \frac{6+6-1}{2 * \sqrt{6} * \sqrt{6}} \simeq .92 \\
\theta &\simeq 22^\circ
\end{aligned}$$

This angle has been multiplied by two in Figure 3 to facilitate the visualization on a half-plane instead of a quadrant.

The measure shows that the concept matching is acceptable, albeit not perfect. The lack of procedural knowledge for the *guideline* concept implies that the diagrams might not follow the prescribed format appropriately. The main concepts are, however, correctly covered, which confirms that this task should not present a knowledge challenge.

4.2. Unacceptable Match

The task *Test Cases Redaction* shown in Figure 3 is in the inacceptable zone. This task builds test cases from the test plan and the architectural models for the application. Formula 8 presents the mismatch measure, where C1 is *testing technique*, C2 is *programming language*, C3 is *programming technique*, C4 is *modeling language* and C5 is *guideline*. Programming is a critical concept here because the test cases are written in the programming language of the application. Good tests must also be tailored to the application itself, requiring the primary role to read and understand the code.

The resource assigned to the role for the *Test Cases Redaction* task was not a competent programmer. This creates a serious issue for this task, which will probably create poor quality test cases, if anything at all. This problem is not obvious to the process designer as processes focus on the flow of data, and not on the knowledge required to handle this data.

4.3. Discussion on the Examples

These two examples show the risks faced by the project manager. An example presented by Jensen and Scacchi [18] shows that processes designed with only workflows in mind can overlook critical details. In their case, a process defined in a rich hypermedia presentation overlooks the workload of the Release Manager, resulting in a process bottleneck. The authors manage to find the problem through process simulation, but it could also be found through knowledge analysis. The Release Manager is required to produce many different documents but is not supported in this work. This will results in long delays between releases, as the Release Manager must acquire all required knowledge prior to producing each document.

5. Concluding Remarks and Future Work

This paper proposes a SPEM extension describing a new approach to concept mismatch identification in software process development. This approach enables the visualization of knowledge discrepancies in software process modeling. Such visualization provides new insights into key practices from the knowledge management viewpoint. An ontological approach coordinates knowledge representation in each SPEM element by linking the process model to one or more ontologies, according to the level of abstraction needed.

The concept matching indicator presents the state of the process model in terms of knowledge and the degree of deviation of each task within the process. This indicator could lead to more informed decision making; for example, in the recruitment of new competencies or the addition of more roles to support the primary role performing the task at hand. As a result, the knowledge-oriented view complements the

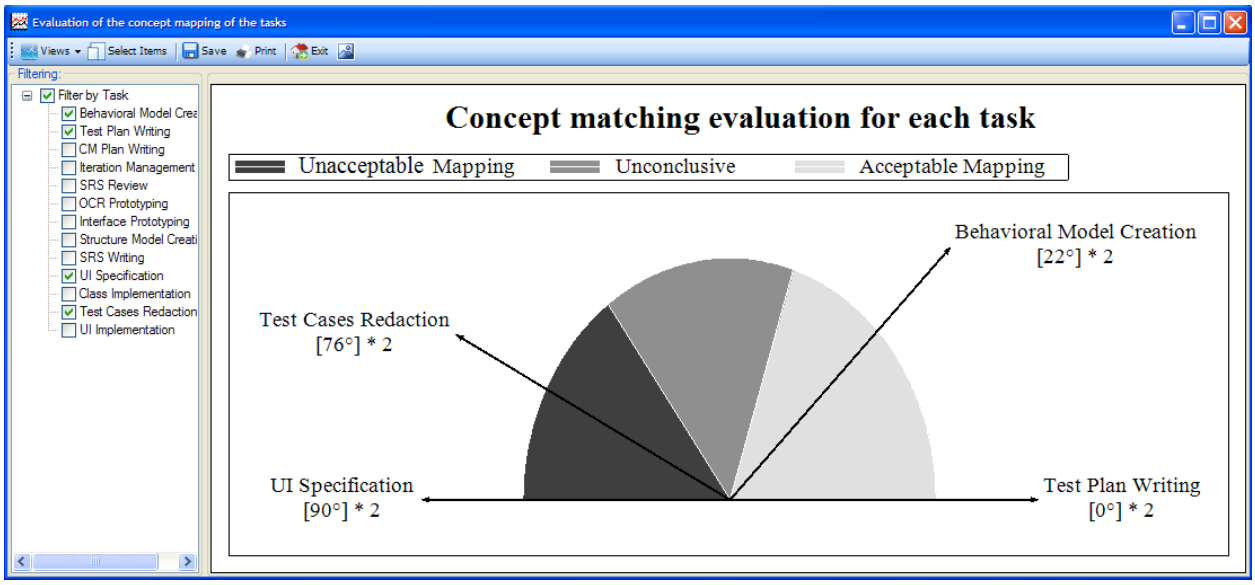

Figure 3. An example of the concept matching indicator

activity-oriented view, thereby fostering a better understanding of complex processes.

Future work will focus on organizational methods that support knowledge creation and propagation related to knowledge flows among software process models. The goal will be to study the propagation of knowledge throughout all the phases of the software process.

It will also seek ways to better integrate the expressivity of common ontological languages like RDF and OWL into our mismatch identification mechanism. This would expand our current tree-structured approach to resemble more flexible structures like the OWL-based SKOS structures.

Acknowledgment

We thank Olivier Gendreau for providing the project data used for our post-analysis. This work was partly supported by the "Fonds de Recherche sur la Nature et les Technologies" council of Québec (FQRNT) under Grant 127037 and NSERC grants A0141 and 361163.

References

[1] P.-N. Robillard, "The role of knowledge in software development," *Communications of the ACM*, Vol. 42, No. 1, January 1999, pp. 87–93.

[2] OMG, "Software & systems process engineering meta-model specification version 2.0," http://www.omg.org/spec/SPEM/2.0/, 2008, accessed: 10/Aug/2012.

[3] N. Kerzazi and P.-N. Robillard, "Multi-perspective software process modeling," in *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010)*, 2010.

[4] R. Martinho, D. Domingos, and J. Varajao, "Concept maps for the modelling of controlled flexibility in software processes," *IEICE Transactions on Information and Systems*, Vol. E93-D, No. 8, August 2010, pp. 2190–2197.

[5] T. H. Davenport and L. L. Prusak, *Working Knowledge: How Organizations Manage What They Know.* Harvard Business School Press, 1998.

[6] T. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* New York: Oxford University Press, 1995.

[7] J. D. Novak and A. J. Cañas, "The theory underlying concept maps and how to construct them," http://cmap.ihmc.us/publications/researchpapers/theorycmaps/theoryunderlyingconceptmaps.htm, 2000, accessed: 10/Aug/2012.

[8] D. P. Asubel, *The psychology of meaningful verbal learning.* New York: Grune & Stratton, 1963.

[9] Y. Wang, "On concept algebra and knowl-

edge representation," in *5th IEEE International Conference on Cognitive Informatics*, 2006, pp. 320–231.

[10] N. F. Noy and D. L. McGuiness, "A guide to creating your first ontology," Stanford University, Technical Report SMI-2001-0880, 2001.

[11] J. R. Anderson, M. Matessa, and C. Lebiere, "ACT-R: a theory of higher level cognition and its relation to visual attention," *Human Computer Interaction*, Vol. 12, No. 4, 1997, pp. 439–462.

[12] F. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Inf. Softw. Technol.*, Vol. 50, No. 11, October 2008, pp. 1055–1068.

[13] J. R. Anderson, *The Architecture of Cognition*. Harvard University Press, 1983.

[14] N. Anquetil, "Software maintenance seen as a knowledge management issue," *Information and Software Technology*, Vol. 49, No. 5, May 2007, pp. 515–529.

[15] N. Ferreira and R. J. Machado, "An ontology-based approach to model-driven software product lines," in *4th International Conference on Software Engineering Advances (ICSEA 2009)*, 2009, pp. 559–564.

[16] J. T. Nosek and I. Roth, "A comparison of formal knowledge representation schemes as communication tools; predicate logic vs semantic network," *International Journal of Man-Machine Studies*, Vol. 33, No. 2, August 1990, pp. 227–239.

[17] G. Salton and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, Vol. 18, No. 11, November 1975, pp. 613–620.

[18] C. Jensen and W. Scacchi, "Experience in discovering, modeling, and reenacting open source software development processes," in *International Software Process Workshop*, 2005.

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 35–44, DOI 10.5277/e-Inf130104

Reusable Object-Oriented Model

Jaroslav Žáček*, František Huňka*
*Faculty of Science, University of Ostrava
jaroslav.zacek@osu.cz, frantisek.hunka@osu.cz

Abstract

This paper analysis approaches and possibilities of executive model aimed to MDA approach. The second part of the article proposes guideline to create executive model, describes basic interactions to object oriented approach and shows possibilities of creating a core of executable model in Java programming language. Annotations are used for executive model object extension. Reflection concept is used for model execution and synchronization provides extended Petri net formalism defined in [1]. The model has been tested on LFLC software package developed by IRAFM, University of Ostrava to prove the whole concept.

1. Introduction

In present days model transformations in object-oriented programming are focused to speed and automation. The criteria for transformation are concrete programming language, model expressivity and domain usability. In addition the elevation of abstraction should be applied to make modeling easy and simple. Main advantages of this approach are noticeable during initial analysis of application or when user needs to automate some processing. During key requirement identification the higher abstraction level is needed. Reducing model abstraction concretizes this initial design with transformations. Transformation ends on source code level and model becomes platform dependent. But in any time the user can transform model to higher abstraction level and make necessary changes. All these tasks can be done using automated tools and changes are applied on lower source code level. This approach is very useful in agile programming methodologies and enables very fast model changes. One option is to divide models to different levels of abstraction and make a transformation between them. Model transformation process is described in [2] specifications and it is know as a Model-driven architecture (MDA). MDA is a registered trademark of Object Management group. The MDA architecture was established in 2001. A lot of transformation tools for platform independent model (PIM) to platform specific model (PSM) were developed since 2001. Tools allows to transfer abstract model to concrete using with technologies such as Web Services, EJB, XML/SOAP, CSharp, CORBA and others. In addition another standard established in the past such as MetaObject Facility (MOF), Unified Modelling Language (UML),Common Warehouse Metamodel (CWM) and XML Meta Interchange (XMI) are available for MDA support. MDA architecture consists of 4 layers specified as a M0 – M3 and every layer in this specification represents a different level of abstraction. MOF is used for initial domain identification. MOF is specified as a M3 layer in a MDA specification. This layer is a domain specific language, which is used for metamodel description. By this language user can describe M2 lower layer. We can consider UML as an object-oriented metamodel and Web Services or Petri's nets as a non-object-oriented metamodel. Models based on MDA architectures are not focused on model execution. These models are focused on platform

independent model transformation to platform specific model and changing level of abstraction.

Executable UML (also known as xUML or xtUML) is a part of UML specification and aimed to execution compared to regular UML diagram and offers needed standard extension for execution.

Executable UML is defined by these elements:

- Class diagram – defines classes and interactions between their associations for the domain
- Statechart diagram – defines the states, activities and state transitions for a class instances
- Domain chart – describes a modeled domain and relations to other domains
- Action language – defines the actions or operations that perform processing on elements

In fact the Executable UML is an extension to MDA platform and enables making executive models on M1 level from elements described above. An executive model on a higher level of abstraction is created and this model is transformed to programming language source code, mostly 3rd generation programming language.

A framework called M3 action has been developed to make executive modeling easy. This framework has been transformed to open-source project called MXF (Model eXecution Framework). Framework extends model with so called action scripts, which express model execution semantic. By these extensions user is able to change model quickly without any implementation or compilation.

2. Problem formulation

Basic formulation of executive modeling has been described in introduction. As a context of problem we consider MDA architecture on Fig. 1. A bottom layer contains data and is an instance of M1 layer, which creates a model. There is no execution on M0 layer because M0 contains data with no context and therefore higher abstraction to express interactions. Interaction between data is realized on M1 layer, where the classes and their relationships are described. These relationships realize method calling. Fast relationship changing is suitable for modeling. By the thought of changing relationship means change any method calling in any object in the model. Ideally user is able to change relationships and inner class attributes during simulation.

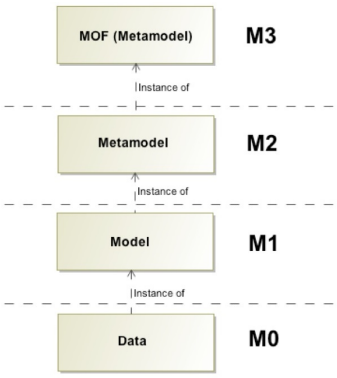

Figure 1. MDA architecture

This execution approach is usually realized on M1 level, which is closed to platform independent model. User can examine classes and their attributes state, make a direct relationship to another class, step the simulation process and ideally read class values in the real time.

2.1. The MDA

The MDA architecture introduced by OMG group was developed to support model-first software development. At first a very abstract model is created, then model is transformed to lower levels of abstraction. Transformations end when model is suitable to generate application source skeleton in corresponding programming language. Automated tools to speed up the process and reduce errors caused by writing code by programmer are available and support this approach.

Model doesn't concern execution and ensures just metadata reflection of workflow. According to [2] MDA is defined by these points:

- Models
- Abstraction
- Platform
- Model Transformation
- The MDA value proposition

MDA specification defines model as a formal specification of functions, structure and system behavior. UML has been chosen as formalism.

According to OMG definition the source code can be concern as a model because this code has a formal specification (all code structures has an exact semantic) and models real machine code, which is available as a program language transformation by compiler or interpreter. This point of view is not interested for object-oriented approach and therefore this model will be considered as a UML model.

Referential model for open distributed processing (MR-ODP) is marked as a suppression of irrelevant detail according to ISO 10746-2 [3]. Model with a high level of abstraction has naturally less detail a posteriori to realization than model with a lower level of abstraction. MDA has been created to start development on a higher level of abstraction and then transform created model to lower level of abstraction until source code is generated. Therefore model drives entire software architecture development.

2.2. Model transformation

MDA generates source code by model transformation. Initial model is platform independent with higher level of abstraction and determined by following points:

- Represents business functionality not tight to technological platform.
- s a detailed model (mostly UML).
- Independent on programming language or operating system.
- Creates baseline to platform-specific model.

PIM is transformed to platform-specified model (PSM) and is adapted to use with target platform. PSM model includes information about business platform and creates PIM mapping to target platform, creates source code skeleton and associated artifacts. As an artifact we can consider deployment descriptor, documentation and build files.

This description implies that we can define a PIM, which can be reusable for different platforms, appropriate PIM to PSM mapping and PSM to source code compiler to target platform as well. In additional this process can by automatized by tools. This transformation is one way only. If the change on lower PSM layer is realized this change cannot be applied on a higher levels in automate process. However, this change could be in a direct conflict with initial modeled purpose. Conversion between PIM to PSM model cannot be realized fully automatically. For example tools cannot determine if account must be marked as an entity EJB or session EJB during the translation.

2.3. The MDA Value proposition

Programming language is an instrument to executive model expressed in UML. This fact has been considered as a disadvantage of model transformation because by this transformation model becomes platform dependent on operating system or specific programming language. Programming language lifetime is limited and when new programming language becomes in use old source code is become useless and must be transferred. Presently using platform independent on operating system approach minimizes the risk of boundedness source code to platform. Using Java technology in these days minimizes boundedness risk. Company's processes are changing and PIM must respond to these changes. The MDA advantage is to preserve high-level views to solve problems – PIM.

2.4. The MDA Execution

In original MDA architecture design was no execution at all. Modeling starts at higher layer and by concretizing model and decomposing (model transformation) the new code is generated. Generated code contains class skeleton. Function interactions between classes are represented by UML relationships only and class itself carries no executive information, instantiation approach or input and output methods. Main disadvantage of this approach is that model cannot use components developed before and model cannot be executed and debugged. PSM to PIM transformation can be made from class diagram (low model view), but this transformation is difficult, cannot be done automatically and for right model identification archetype patterns [2] must be used. To make MDA architecture running automatically an Executable UML extension must be applied.

2.5. M3 Action – Model Execution Framework

M3 action, mostly known as a MXF, is a project focused on executive modeling on a higher level of abstraction (M3). A new language has been defined to describe interactions between elements [4]. Language is based on UML Actions/Activities. From executive point of view, a more abstraction view is available compare to Executable UML. MXF and Executable UML cannot change level of abstraction and creates executable models on a single layer. Metaobject instantiation is performed in M3 abstraction level; therefore tool cannot identify a design pattern of the implementation. Compare to UML the MXF supports aspect-oriented programming due to M3 abstraction level.

One of the main goals of object-oriented programming is reusing components. In all approaches described above there is no mechanism to integrate reusable components to model or make model with reusable components. Approaches discuss creating class skeleton of model in programming language with no direct execution. Model created that way cannot be debugged without changing source code and add some new functionality. MDA architecture is able to generate model from bottom to up (elevate level of abstraction) by using the archetype patterns, but this model lost executive ability by perform this transformation. Executable UML tries to minimize MDA disadvantage by adding more information to object interaction in the model. By applying these techniques an executable model with higher abstraction level from reusable components cannot be created.

3. Defining a new modeling approach

MDA, Executive UML and MXF don't include the requirements to executive model:

- Create model form reusable components.
- Concerning design patterns.
- Flexible change when component is replaced.
- Function and debugging with no code compilation.
- Change the level of abstraction.

These requirements can be realized with minimal generality reduction by extension of object metamodel and applying a reflection tool.

3.1. Reflection

Reflection as a term in information science means ability to read and change program structure and behavior during the program running. Considering to object-oriented programming approach, reflection means ability to read and change object attributes, read and execute the object methods, passing calling results and instantiate new objects. Generally the reflection is able to read object metamodel during program running without changing any object attributes. Reflection is widely used with Smalltalk programming language and scripting languages. Reflection can be used as a universal tool to make object persistent [5] or to generate project documentation.

Reflection enables creating a new object instance entered by name during program running. Following source codes are in Java programming language, but same function can be done with .NET platform and languages defined under Common Language Specification. Basically there are two requirements to programming languages:

- Ability to read object metadata and work with them as a metamodel (object self-identification).
- Some tool to enable object metamodel extension.

The metamodel that carries information about class must be discovered before instantiation.

Fig. 2. shows a representation of metamodel reference. That reference has been found during program running by providing his name – String data type. Execution wrapper is a standalone class. Inner attribute saves metamodel references and instantiated object. For every object is created his instance, special cases as a Library Class are covered by metamodel extension explained in 3.2.2.

At first a constructor must be found to instantiate a class. A simple model has been created for model testing purpose. Model is limited to

```
public boolean setReflectObjectByName(String name)
        throws ClassNotFoundException, SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        IllegalAccessException, InvocationTargetException,
        InstantiationException {
    boolean instanceLimit = false;
    String instanceMethod = "";
    int pool = -1;
    this.reflectClass = Class.forName(name);
    Annotation[] annotations = this.reflectClass.getAnnotations();
    for (Annotation annotation : annotations) {
        if (annotation instanceof ModelSupport) {
            ModelSupport type = (ModelSupport) annotation;
            if (type.designPattern().equals("Singleton")) {
                instanceLimit = true;
                instanceMethod = type.instanceMethod();
                pool = 1;
            } else if (type.designPattern().equals("Pool")) {
                instanceLimit = true;
                instanceMethod = type.instanceMethod();
                pool = type.pool();
            } else {
                instanceLimit = false;
            }
        }
    }
    if (instanceLimit) {
        return createInstanceLimit(instanceMethod, pool);
    }
    return createInstance();
}
```

Figure 2. Pointer to object metamodel

non-parametric constructor. During instantiation the metamodel is searched and first constructor is called. Result of instantiation is saved to class realizing execution. A method to create instance with no instantiation number limit is described in Fig. 3.

More complex instantiation method is extended by instance count parameter and factory method name. Factory method specification is presented in virtue of factory method name inconsistency [6].

3.2. Analyzing class

Reflection can read all object metadata, all inner attributes, methods, input parameters and return value can be identified. A new class has been created (Fig. 4.) to metamodel verification. Reflection is applied to find the metamodel and all methods are called one by one. Metamodel contains a list of all methods including methods marked as a private by modifier.

Discovered metamodel will be used as an input information to create a graphic model representation. In this graphic representation an order of method calling can be changed if all input attributes and return values have same type.

Eleven modifiers are defined by Java programming language. Modifiers can be characterized as a possible access to objects. In Fig. 3. the modifier is set to private therefore a violation of object-oriented programming is occurred. But it isn't a mistake from instantiation point of view. According to Library Class design pattern, the constructor is defined as an empty constructor therefore Library Class instantiation doesn't change inner state of object. Other classes using a factory method to instantiation requires at least a metamodel extension for factory method identification. Change can be done on graphic model representation level as well. According to reflection all identifiers can be changed, which gives user powerful tool to make changes during the program/model running.

```
private boolean createInstance() throws SecurityException,
        NoSuchMethodException, IllegalArgumentException,
        IllegalAccessException, InvocationTargetException,
        InstantiationException {

    Constructor[] constructors = this.reflectClass
            .getDeclaredConstructors();
    for (Constructor constructor : constructors) {
        if (!Modifier.isPublic(constructor.getModifiers())) {
            constructor.setAccessible(true);
        }
        this.reflectObject = this.reflectClass.newInstance();
        break;
    }
    return true;
}
```

Figure 3. Basic instantiation method

```
public boolean invokeAllMethods() throws IllegalArgumentException, IllegalAccessException,
                                                                    InvocationTargetException {
    Method[] allMethods =  this.reflectClass.getDeclaredMethods();
    for (int i=0; i<allMethods.length; i++) {
        Method oneMethod = allMethods[i];
        Class<Type>[] parameters = (Class<Type>[]) oneMethod.getParameterTypes();
        Object[] defaultValues = new Object[parameters.length];
        for (int j=0; j<parameters.length; j++) {
            Type type = parameters[j];
            defaultValues[j] = getDefaultType(type);
        }
        Object returnObj = oneMethod.invoke(this.reflectObject, defaultValues);
        if (returnObj.getClass().isArray())
            System.out.println("Calling method according to type, return value is array.");
        else
            System.out.println("Calling method according to type, return value is " + returnObj.toString());
    }
    return true;
}
```

Figure 4. Invoking methods

If reflection is used to create executive model a question of speed of entire executive lifecycle needs to be consider. Supporting class ensures not just initial instantiation but calling methods and passing parameters as well. According to some sources reflection API is slow. Confirmed by [5] this affirmation is not based on true. By application of Amdahl's law a formula is derived:

$$\text{Slowdown} = \frac{\text{Rtime} + \text{Work}}{\text{Ntime} + \text{Work}}$$

where Rtime is a micro benchmark measurement of a reflection solution and Ntime is a micro benchmark measurement for nonreflective solution. Work is a relative amount and can be interpreted as a Work = Ntime * x , where x is a factor of scaling, which determines time spend with other things. By substitution a formula to slowdown can be derived:

$$\text{Slowdown} = \frac{\frac{\text{Rtime}}{\text{Ntime}} + x}{1 + x}$$

This interpretation of Amdahl's law enables to set referential unit of performance. Rtime/Ntime ratio should be about the same for processors no matter the speed at which the clock is running. After value substitution of instantiation dynamic proxy the ration is equal to 329.4 and slowdown is about 1900% when work is less than 17 times NTime. This numbers seem high but to print "Hello World!" in Java programming language the value of work is equals to 36,000 times Ntime, which a slowdown is under 1%. Therefore there is no noticeable slowdown if reflection technique is applied.

3.3. Class metamodel

According to [7] a metamodel is a domain-specific language oriented towards the representation of software development methodologies and endeavours. After adjusting to class diagram metamodel we can say that metamodeling is an ability to express interactions between classes from metamodel – inner object state. Metamodeling is the act and science of engineering metamodels. Basic metamodel contains information necessary to class representation in concrete programming language.

Two approaches can be use to get metamodel. Model can be obtained from descriptors made before which are tight with created class. This form of implementation is very simple, however descriptor maintenance becomes difficult. When descriptors are defined in high amount the maintenance becomes confusing. If the class doesn't contain descriptors, it cannot be used for metamodel purpose. This type of approach is applied in object-relation mapping known as a Hibernate project.

Second option is use a reflection and read entire object metamodel. This information is obtained during program running and therefore enables dynamic 3rd part library linking with no additional library changes. When class name is provided the reflection interface can read all class attributes, methods, return values and modifiers and pass these values to process on a higher level, typically GUI. In some cases detail information must be known to use class metamodeling. Basic metamodel is not sufficient therefore a new tool for user metamodel extension needs to be found. Reflection must be able to use these extensions during object instantiation and modeling. Annotations are a quiet suitable for user metamodel extension. Annotations are special type of syntactic metadata, which can be add to class source code and extend metamodel expressivity. Metamodel expressivity extension is shown on Fig. 5.

Interface on Fig. 5. is implemented by a class and extends user description part. Reflection allows reading these values and directly decides during program running. In this case a definition of instance is presented. Class carries information about instantiation limit in the metamodel and solves interaction between design patterns and a class model.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)

public @interface ModelSupport {
    public String designPattern();
    public String instanceMethod();
    public int pool();
```

Figure 5. Metamodel extension

A model can be realized based on previous recommendations. For a low price – less generality – model solves all problem points identified above. Model will support to plug 3rd part components, design patterns will be instantiated in a right way. Model is executable in any time and brings immediately operational picture of modeling reality. This model is realized by reflection as a supporting mechanism for execution and debugging during program running. Reflection makes model free to use 3rd part components. Model makes instantiation of these components and other classes, calls corresponding methods defined in model and passes parameters.

3.4. Basic entity view

Graphics representation of basic model scheme suggests Fig. 6. Final list of atomic classes are available. This list represents single classes but relationships are simplified from methods to object links. In simplified model an antecedent has only one consequent and antecedent pass result process directly to consequent. Reflection realizes a passes of result and instantiation in right way with interaction to design patterns. Design pattern accuracy ensures the extended metamodel. Model input and output is defined. Every element in model has only one input and one output.

More complex metamodel presents Fig. 7., which is extension of basic model. This model is closer to reality because some entities presented in the model has more then one input. Output is limited to one because of programming language limitation. A synchronization problem occurs if method has more then one input. In this case we

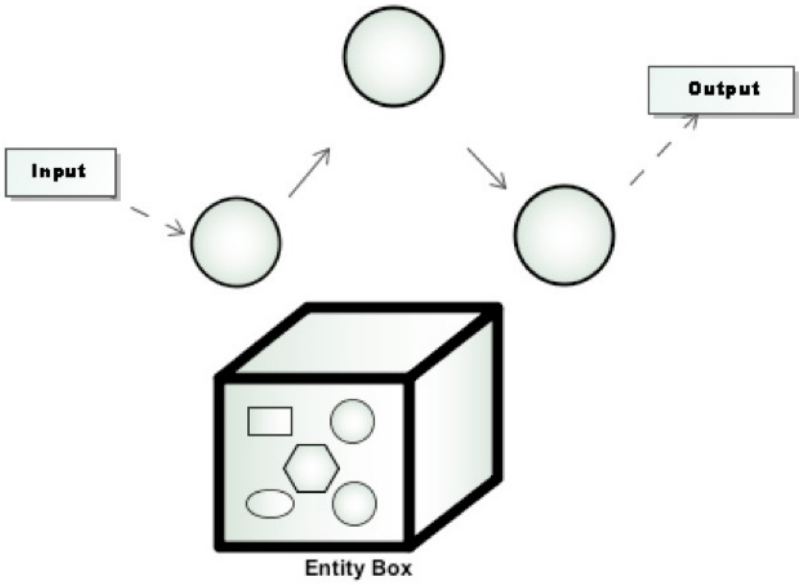

Figure 6. Basic model

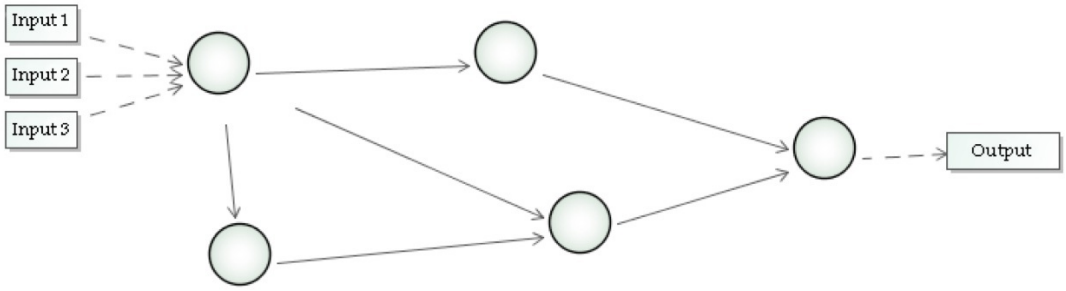

Figure 7. Extended model

must apply object-valued Petri net introduced in [1].

3.5. Class view

For an executive model representation based on reflection and annotations is more useful to create a class view. This class view shows Fig. 8., where every entity from Fig. 7. is transformed into the class. UML notation is chosen willfully because of wide using in practice. Every class contains an internal and external method. Internal methods are marked with private modifier, external with public modifier. Same approach is applied to attributes. Modeling process starts when user enters initial values and the smallest stem in simulation is one executed method. An internal state of object is changed during method execution or when the return value is generated. Returned value is passed to the next class. User can observe every object attribute and read return value after every step of execution. This feature enables reflection. User can also change interactions between objects during program running. User is able to use internal methods by changing modifiers. Internal state of the object can be edited as well. These features give user ability to create the executive model with no source code writing. This can be advantageous when result cannot be predicted but result might influent consequent components – chaining calculation. Nowadays many examples can be found. User gets possibility to create more complex structures and debug these structures after every step with no compiling. Model allows plugging some new classes during simulation.

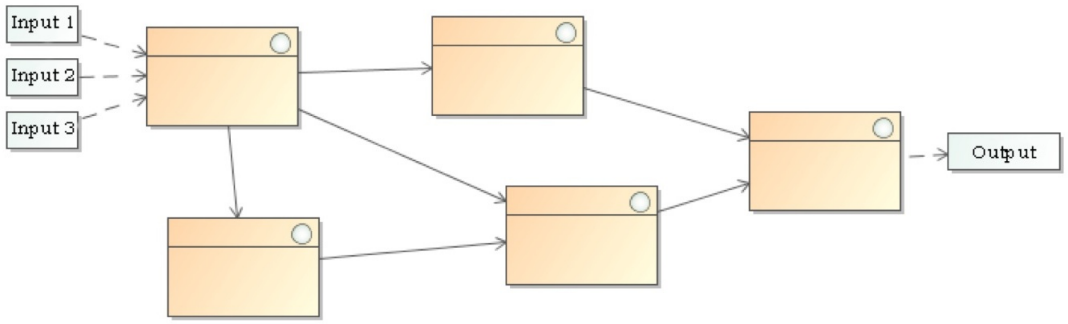

Figure 8. Extended model

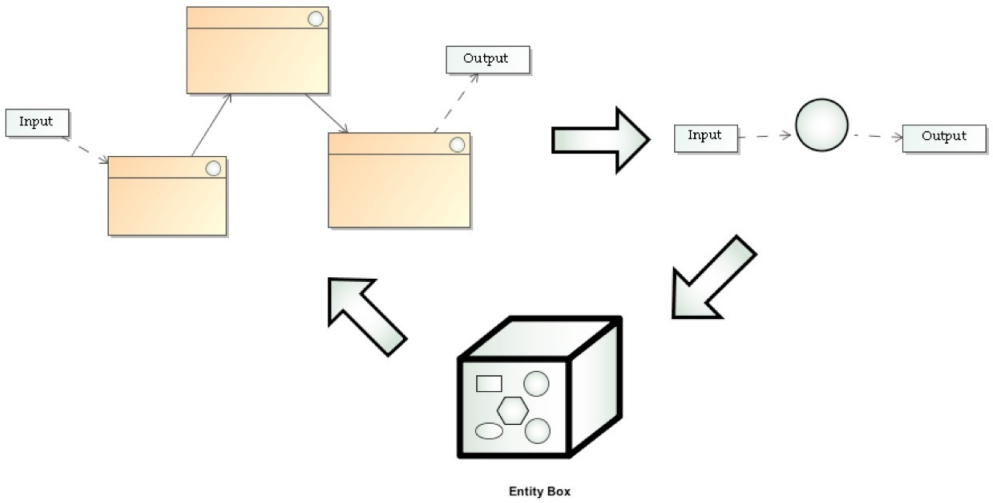

Figure 9. Level of abstraction

Metamodel, read by reflection, allows creating graphical object representation in a model. Final relationships between classes can be saved by structured XML document. XML assigns unique identifiers to classes, defines inputs and outputs and mutual return value passing.

3.6. Elevate level of abstraction

Very important model feature is ability of elevate model of abstraction. In strict metamodeling framework an instance-of operator is allowed only within layers in a same linguistic level. However if we consider ontological level we can use instance-of operator on any layer. By linking on different layers new entities arises. These entities describe [7], namely Clabject (class-object) and Powertypes. On Fig. 9. is shown a mechanism to elevate level of executive model abstraction. Model created by user consists of several classes and interactions between them. Classes are part of the entity box. This executive model is transformed to single entity after debugging and testing and carries description of significance and defines input and output point. Entity becomes a part of entity box as a single atomic element and therefore is available to future modeling of executive models. User can edit created entity and modify internal relationships or whole classes.

4. Conclusion

This paper introduces a practical proposal of new executive modeling approach introduced on [8]. First part of the paper defines problem domain and related approaches to create executive models. Following paragraph describes a reflection

application to executive model, which enables component integration. All proposals are programmed and integrated with Java programming language. Reflection can slow model processing therefore an Amdahl's law is applied to prove that there is no significant computer processing slowdown. Created model has been verified on LFLC software and brought a significant speedup during changing inferential mechanism. By implementing object-valued Petri net formalism introduced in [1] synchronization problems in complex models has been managed well. Paragraph 3.6 clarifies a possibility of elevate level of abstraction of the new executive model where the future work will continue.

Acknowledgement

This paper is supported by IGA no. 6141, Faculty of Science, University of Ostrava.

References

[1] J. Žáček and F. Huňka, "Object model synchronization based on petri net," in *Mendel 2011: 17th International Conference on Soft Computing, June 15-17, 2011*, R. Matousek, Ed. Brno: Brno University of Technology, 2011, pp. 523–527.

[2] J. Arlow and I. Neustadt, *Enterprise patterns and MDA: building better software with archetype patterns and UML*. Boston: Addison-Wesley, 2004.

[3] "ISO/IEC 10746-2:1996 information technology – open distributed processing – reference model: Foundations," 1996. [Online]. http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[4] M. Soden, "Operational semantics for MOF metamodels." [Online]. http://www.metamodels.de/docs/tutorial_draft_v1.pdf

[5] I. R. Forman and N. Forman, *Java reflection in action*. Greenwich, Conn.; London: Manning; Pearson Education, 2005.

[6] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley, 1995.

[7] C. A. González Pérez and B. Henderson-Sellers, *Metamodelling for software engineering*. Chichester, UK; Hoboken, NJ: John Wiley, 2008.

[8] J. Žáček and F. Huňka, "CEM: class executing modelling," *Procedia Computer Science*, Vol. 3, 2011, pp. 1597–1601. [Online]. http://www.sciencedirect.com/science/article/pii/S1877050911000561

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 45–52, DOI 10.5277/e-Inf130105

From Principles to Details: Integrated Framework for Architecture Modelling of Large Scale Software Systems

Andrzej Zalewski*, Szymon Kijas*

*Institute of Automatic Control and Computational Engineering, Warsaw University of Technology

a.zalewski@elka.pw.edu.pl, s.kijas@elka.pw.edu.pl

Abstract

There exist numerous models of software architecture (box models, ADL's, UML, architectural decisions), architecture modelling frameworks (views, enterprise architecture frameworks) and even standards recommending practice for the architectural description. We show in this paper, that there is still a gap between these rather abstract frameworks/standards and existing architecture models. Frameworks and standards define what should be modelled rather than which models should be used and how these models are related to each other. We intend to prove that a less abstract modelling framework is needed for the effective modelling of large scale software intensive systems. It should provide a more precise guidance kinds of models to be employed and how they should relate to each other. The paper defines principles that can serve as base for an integrated model. Finally, structure of such a model has been proposed. It comprises three layers: the upper one – architectural policy – reflects corporate policy and strategies in architectural terms, the middle one –system organisation pattern – represents the core structural concepts and their rationale at a given level of scope, the lower one contains detailed architecture models. Architectural decisions play an important role here: they model the core architectural concepts explaining detailed models as well as organise the entire integrated model and the relations between its submodels.

1. Introduction

Large scale software intensive systems are built to serve country- or worldwide organisations employing thousands of users. They span across the organisation's entities and locations often needing to cope with distributed data storage management and processing. The research presented in this paper has been motivated by the lack of effective approaches to the modelling of architecture of such systems. This is caused by the gap existing between the modelling frameworks and software architecture models.

The modelling frameworks classify information describing system structure rather than indicate precisely how this information should be represented with appropriate architecture models. On the other hand, existing models and modelling approaches usually represent selected viewpoint on system architecture, however it is not clear how these different models should be integrated to create an effective architecture modelling engine.

The research envisaged in this paper is aimed at integrating various models and modelling approaches into a uniform, integrated framework

providing a precise guidance on employed models and structure of their relations. The rest of the paper is organised as follows: state-of-the-art in architecture modelling is analysed in section 2, section 3 presents the basic rulesguiding the design of an integrated architecture model, section 4 contains a proposition of such an integrated architecture model illustrated with a real world example, concept summary and range of further research comprise section 5.

2. State-of-the-art in Architecture Modelling

Architectural description of software-intensive systems seems to be a well established practice. It is defined by IEEE Std 1471-2000 and draft standard ISO/IEC 42010:2007 [1]. These standards comprise the most important concepts of the architecture genre as: stakeholders' viewpoints, stakeholders' concerns, architecture views [2] and even architecture rationale [3]. However, these standards do not indicate how architecture modelling should be done in practice, i.e. what kind of models shall be used, how such a suite of architecture models should be organised, verified etc.

We argue that this challenge have not been met yet at least in case of the modelling of large scale software systems. The above standards are rather of a declarative than of an imperative style.

Enterprise architecture frameworks as Zachman Framework [4,5] or The Open Group Architecture Framework (TOGAF) [6] belong to the same declarative genre: they classify information describing architecture neither indicating how this information should be represented and what models shall be applied nor how different classes of information are interwoven and interact with each other.

On the other hand, there is a large variety of architecture description languages (ADL). ADL's as ACME, Wright, Aesop, UniCona and xADL (for full reference see [7]) have not reached the level of industrial application maturity. In contrast: Unified Modeling Language (UML) plays a role of an industrial standard, however, because of its origin, it is rather recognized as a set of models of software than system architecture. No wonder, UML is mainly applied in the context of 4+1 views [2] of software architecture (logical view, process view, physical view, development view, scenarios).

Architectural decisions [8, 9] are often perceived as another wave in architecture modelling – compare "third epiphany" in [3]. The idea that systems architecture, as every design, results from a set of decisions seems to be strongly appealing to both engineers and scientists. Architectural decisions can potentially represent any architectural concept belonging to any architectural view [3]. On the contrary to the other architecture models as UML or ADL's architectural decisions help to capture design rationale, which is a part of tacit knowledge which usually evaporates as soon as design is ready or as architect is gone. This ability to capture design intent is perceived as the most important advantage that architectural decisions provide.

However, hopes that architectural decisions alone will become an effective model of software architecture seem to be unfounded, especially in case of large scale software systems. The fundamental limitations of this modelling approach have been investigated in our former paper [10]. Architectural decisions are represented as text records, sometimes accompanied with illustrating diagrams [11]. The limitations of textual models are well-known in the genre of software engineering. Therefore, sets of hundreds of architectural decisions necessary to sufficiently represent architecture of a large system, are difficult to comprehend, analyse, verify and ensure completeness and consistency or even just to navigate through them.

This creates a substantial risk that modelling approaches based only on architectural decisions will collapse under their own weight as they create complexity of their own rather than helping to control complexity of system architecture. The effort and cost necessary to create and maintain such a large set of architectural decisions can discourage engineers and managers from using them at all. The existence of standards (e.g. IEEE-1471) and commercial modelling frame-

works (e.g. TOGAF) should indicate that architecture modelling have matured to the industrial application. However, the gap between the declarative standards/frameworks and concrete architecture models still exists. For this reason, newer frameworks and models still emerge – compare recent developments: architectural decisions [12], recent versions of TOGAF [6] or Archimate notation and modelling approach [13] both promoted by The Open Group.

The challenge is to make an efficient modelling framework out of the existing models and frameworks (at least parts of them), in accordance with existing standards.

3. The Basic Rules for the Design of an Integrated Model of System Architecture

The observations presented below are supposed to exploit the advantages of the models and approaches presented above, while trying to minimise their drawbacks. They are aimed at providing foundation for integrated models of software architecture.

1. **Design rationale should be captured only for the most important design elements. Hence, architectural decisions should express only the core design concepts that are necessary to comprehend the structure of a given design component. They should by no means express design details.**
   The value of good system architecture is that it defines a kind of a skeleton defining basic organisation of every system's entity. This skeleton should remain almost unchanged throughout the life time of a given system entity. Here are just two examples of such skeleton-decisions:
   - design pattern (e.g. broker, model-view-controller) defines fundamental design structure of a given software component usually remaining unchanged as long as the component exists;
   - decision that the corporate systems will be integrated at two levels: domain and enterprise: there will be systems (e.g. Enterprise Service Bus – ESB , Business Process Management – BPM) integrating systems belonging to certain domains (e.g. sales, financial management) and a separate system integrating domains at corporate level – provides a structure, to which future system developments have to be tailored.

   Representing these basic structural concepts does not require modelling of all the details with architectural decisions as diagrammatic models are usually more efficient. This should help to overcome the intrinsic limitations of architectural decisions simultaneously making ADL/UML models easier to comprehend.
2. **Models representing the details of systems architecture should be chosen adequately to the class and properties of the modelled system as well as its stakeholders' concerns.**
   This observation is a consequence of the former one: core concepts can effectively be modelled with architectural decisions, while efficient modelling of design details can only be done with appropriately chosen models. Efficient architecture models will be different for different classes of systems – compare e.g. Service Oriented Architecture (SOA) and real-time systems. It is also worth noting that different models can be useful for different stakeholders' concerns – e.g. performance, security, reliability. Hence, the contents of integrated model should be chosen with respect to both the selected class of systems and concerns of architecture stakeholders.
3. **Architectural decisions should explain detailed models**
   Models of systems architecture like ADLs or UML are usually easier to comprehend when their underlying concepts are clearly stated – e.g. it is much easier to analyse a class diagram for model-view-controller component if you know in advance that this pattern has been followed. It is often difficult to deduce such an intent straight from the class diagram itself. Linking architecture decisions with architecture models can make ADLs more ef-

ficient. This is quite an opposite approach to [11], where architectural decisions are illustrated with diagrams. In fact, both possibilities are included in the proposed model.

4. **Architecture should be represented at different levels of detail/scope/view (scope [14]), being useful for different stakeholders**
   The complexity of large scale software system architecture results from the fact that organisation of software systems can be perceived at different levels of details and from the viewpoints of different stakeholders. This is both virtue (helps to cope with systems complexity) and vice (the relations between models at different levels of scope and/or belonging to different views are by no means clear increasing the overall architecture model complexity).
5. **Architectural decisions related to the detailed models can become a kind of an index helping to navigate through them**
   The architectural decisions are a versatile model of system architecture modelling efficiently only its core concepts. As such they can be used to integrate all the models across all the levels of scope, detail and views. If detailed models are appropriately linked to the core architectural decisions they can be used as a kind of an index to the detailed models.
6. **Integrated model should support systems evolution**
   Almost all the software systems are subject to changes throughout their lifetime. Hence, architecture models representing current-state system architecture only are of a limited use nowadays. Mechanisms of capturing changes, presenting model snap-shots for a selected moment of time should accompany an integrated architecture model.
7. **Support alignment of systems architecture with business strategy/policies**
   The need for the alignment with business strategy and policies does not have to be explained. However, it is usually difficult to assess whether architecture of systems or IT products really supports business strategy. Therefore, integrated architecture model shall make such an assessment easier.
8. **Promote and enable validation/verification of one model against other connected ones (especially higher level models).**
   As architectures are modelled at different levels of scope, detail and from different points of view these models can potentially be assessed/verified/validated one against another. This requires that architecture models are appropriately organised and interrelated with each other.

4. The Integrated Model of Large Scale Software System Architecture

The concepts of integrated model of large scale software systems will be illustrated with an example of a real system used in the banking sector. The system presented in fig. 1 has been developed to support the exchange of various kinds of information and documents (claims, direct debits, information concerning accounts moved from one bank to another, etc.) between banks and other institutions (e.g. bailiffs' offices, social security agencies). Additionally, it is used as a fail-over communication channel in case of a failure of the main clearing system. The system generally follows the service oriented architecture scheme providing both www and web services interfaces to its functionality.

The overall structure of the proposed integrated model of software architecture has been presented in fig. 2. It comprises the following tiers:

– **Architectural Policy**: comprises a set of clauses that translate the enterprise strategy and relevant policies to a set of principles shaping the architecture of corporate IT systems. Architectural Policy is represented as a set of clauses being simply text records consisting of the following fields (similar to the architecture principles of TOGAF [6]): a.p. policy rule name, a.p. policy rule, a.p. rule explanation. As Architectural Policy reflects

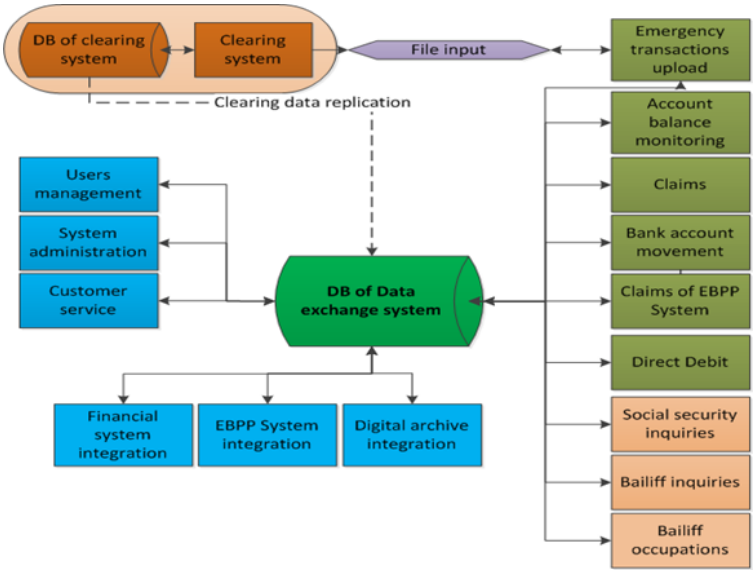

Figure 1. Banking data and document exchange system

enterprise strategy and relevant policies in the architectural categories it can provide criteria for the assessment of the alignment between system architecture and business strategy or enterprise policies.

Our example: if the corporate strategy says that the mission of the company is to provide fully secure services to banking sector, it can be translated into a number of architectural policy clauses – e.g. "all in-coming and out-going data is securely transferred", "all the processing of clients data can be back-traced", "all the users have to be securely authorised before accessing its services". These rules can be used to verify/assess the rules of system organisation pattern (in our example: the first architecture policy clause applies to the rules concerning communication framework selection and data input organisation (see below).

- **System Organisation Pattern**: is set of architectural decisions (called System Organisation Pattern Rules) representing core concepts organising system (this extends the idea presented in [15]) at different levels of scope [14]: enterprise, domain, system/application, component. These basic decisions include but are not limited to:
  - *Decomposition into set of domains/subsystems/applications*: defines organisation of system functionality – from conceptual or business (domain) to technical level (applications/subsystems).

    Our example: the following domains have been defined: Human Resources (HR) Management, Finance Management, Clearing Systems, Digital Signature and Auxiliary Services. The „Data exchange system" belongs to Clearing Systems Domain. It has been divided into 15 subsystems/modules (comp fig. 1)
  - *Geographical and organizational allocation of system entities*: defines both the deployment of system entities in terms of organization's geography and organization structure.

    Our example: The components of „Data exchange system" are supposed to be deployed in central company's data centre, client applications will be provided throughout the company and its clients.
  - *Organisation of data input*: indicates where and how the data is fed into the system.

    Our example: The data will enter/leave the system via central interfaces only: WWW interface, web service interface and file interface.

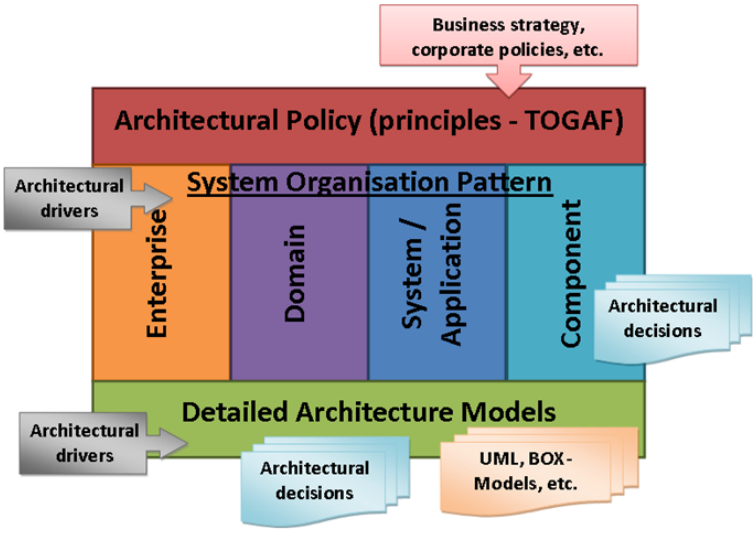

Figure 2. The overall structure of the integrated architecture model for large scale software systems

- *Data storage distribution*: defines how permanent data is distributed among databases;
  Our example: Lack of distributed data (all of data are allocated in one central database) eliminates the need for distributed data storage management and allows for short transactions only.
- *Distributed data storage management*: defines the means of database synchronization, data transmission between remote locations etc.
- *Transaction management*: concerns the selection of mechanisms and/or solutions that are used to ensure transactional processing.
- *Communication framework*: defines communication mechanisms between equivalent system entities (e.g. queuing solutions).
  Our example: subsystems exchange data only via database; clients communicate with the system via SSL channel.
- *Core pattern/style selection*: the selection of design patterns or architectural styles that define the overall structure of a given architecture entity.
  Our example: all the subsystems are supposed to follow three-tier architecture.

These decisions are supposed to be represented as text records as in [9, 16]. Other models can be employed here as they emerge.

- **Detailed Architecture Models**: represent details of system architecture. Obviously the suite of models used at this level has to be tailored to the class and properties of the modelled system. We plan to develop such a model suite for SOA systems as an integral part of our research and a mean of validation of our concepts. Obviously, other suites should also be developed for other domains or classes of systems.

The detailed structure of the contents of the integrated architecture model has been presented in fig. 3. The following properties could be observed from the model presented in fig. 3:

- Rules of system organisation pattern can be related to a number of architectural policy clauses, which should support the assessment of strategic alignment of system architecture;
- A number of rules of system organisation pattern can be applied to a given detailed model, sometimes detailed model can be used to illustrate a given rule;
- System organisation pattern rules belong to one of the levels of scope and describe architectural entities defined at this level;

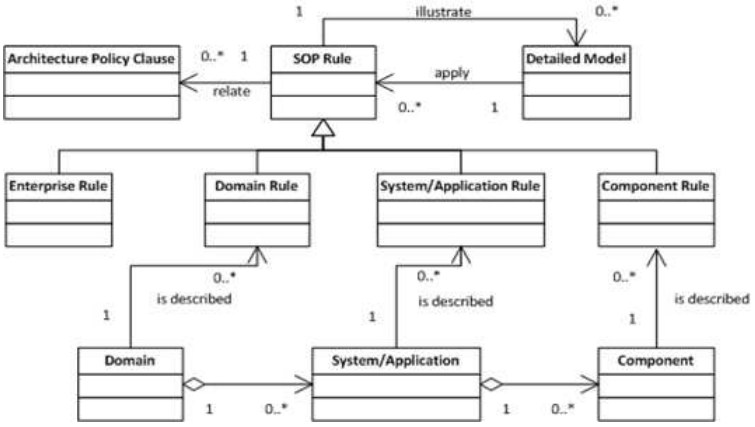

Figure 3. Class diagram illustrating dependencies between model entities

– Association between system organisation pattern rules and detailed models indicates models of a given system entity.

The dependencies that may exist between system organisation pattern rules have not been shown to make the class diagram more legible, however, enterprise level rules, can be applicable to some lower level decisions, especially domain ones, the latter ones to the system/application rules etc. etc. These dependencies can be used to verify whether lower level models are compliant with higher level ones.

5. Summary. Further Research Prospects

The integrated system architecture model presented in section 4 fulfils most of the premises enumerated in section 3. Architectural decisions model only core architectural concepts at a given levels of scope (rule 1, 3, 4), relations established between them and detailed models provide for the indexing of detailed models (rule 5). Alignment with business strategy (rule 7) is ensured with Architectural policy being an integral component of the model.

Further research is needed to:

– Define model suites for different classes of systems and their stakeholders (rule 2) – our research is heading towards identification of such a suite for SOA systems – these are supposed to include at least: BPMN as business process models (at different levels of detail) and LOTOS as a formal model of concurrent processing in Business Processes providing for extended verification capability;
– Enhance integrated model with mechanisms supporting system evolution (rule 6);
– Although model is structurally ready for the assessment of one model against another one, effective analysis/assessment techniques can be developed when full suite of models is defined as well as verified properties are know. This can only be done in the context of a certain genre of software-intensive systems as e.g. service-oriented systems. (rule 8).
– Develop tool support the integrated architecture model.

Acknowledgment

This work was sponsored by the Polish Ministry of Science and Higher Education under grant number 5321/B/T02/2010/39.

References

[1] *Recommended Practice for Architectural Description of Software-Intensive Systems*, http://standards.ieee.org/findstds/standard/1471-2000.html, IEEE Std., 2000.

[2] P. Kruchten, "The 4+1 view model of architecture," *IEEE Software*, Vol. Volume 12, No. 6, 1995, pp. 45–50.

[3] R. C. Philippe Kruchten and J. C. Dueñas, "The decision view's role in software architecture prac-

tice," *IEEE Software*, Vol. Volume 26, No. 2, Mar/Apr 2009, pp. 36–42.

[4] J. Zachman, "Framework for information systems architecture," *IBM Systems Journal*, Vol. Volume 26, No. 3, 1987, pp. 276–292.

[5] ——. (2010) Zachman framework. http://www.address.org/.

[6] The Open Group Architecture Framework (TOGAF). http://www.zifa.com. The Open Group.

[7] A. W. Kamal and P. Avgeriou, "An evaluation of ADLs on modeling patterns for software architecture," *LNCS Springer*, Nov 2007, 3rd International Workshop on Rapid Integration of Software Engineering techniques (RISE).

[8] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," *IEEE Computer Society*, 2005, pp. 109–120, 5thWorking IEEE/IFIP Conference on Software Architecture (WICSA'05).

[9] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," *IEEE Software*, Vol. Volume 22, No. 2, Mar 2005, pp. 19–27.

[10] A. Zalewski and S. Kijas, "Architecture decision-making in support of complexity control," *LNCS Springer*, Vol. Volume 6285, 2010, pp. 501–504.

[11] F. N. Rafael Capilla and J. C. Dueñas, "Modeling and documenting the evolution of architectural design decisions," *IEEE CS Press*, 2007, proc. 2nd Workshop Sharing and Reusing Architectural Knowledge Architecture, Rationale and Design Intent.

[12] P. Kruchten, "An ontology of architectural design decisions," *Rijksuniversiteit Groningen*, Oct 2004, pp. 54–61, in 2nd Groningen Workshop on Software Variability Management.

[13] ArchiMate® 1.0 specification. http://www.opengroup.org/archimate/doc/ts_archimate/.

[14] R. Malan and D. Bredemeyer, "Less is more with minimalist architecture," *IEEE's IT Professional*, Sep/Oct 2002.

[15] A. Zalewski, "Beyond ATAM: Architecture analysis in the development of large scale software systems," *LNCS Springer*, Vol. Volume 4758, Sep 2007, pp. 92–105, first European Conference, ECSA 2007 Aranjuez, Spain.

[16] P. A. Neil B. Harrison and U. Zdun, "Using patterns to capture architectural decisions," *IEEE Software*, Vol. Volume 24, No. 4, Jul/Aug 2007, pp. 38–45.

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 53–64, DOI 10.5277/e-Inf130106

Time Domain Measurement Representation in Computer System Diagnostics and Performance Analysis

Stanisław Wideł*, Jarosław Flak*, Piotr Gaj*

*Faculty of Automatic Control, Electronic and Computer Science, Silesian University of Technology

stanislaw.widel@polsl.pl, jaroslaw.flak@polsl.pl, piotr.gaj@polsl.pl

Abstract

Time analysis is a common approach for testing and detecting methods for the performance analysis of computer systems. In the article it is shown, that measuring and identifying performances based on a benchmark is not sufficient for the proper analysis of the computer systems behavior. The response time of the process is often composed of the execution of many subprocesses or many paths of execution. Under this assumption, it is presented, that both convolution and deconvolution methods can be helpful in obtaining time distributions and modeling of complex processes. In such a modeling the analysis of measurement errors is very important and was taken into consideration. The example of using the methods in buffering process is also discussed.

1. Introduction

Performance analysis and diagnostics of system software is a difficult problem even for simple computer systems. In working systems many processes run simultaneously and influence each other, so time analysis is a complex task. The time analysis includes a number of different approaches. The most important ones are the worst-case time analysis, the performance analysis, and benchmark tests. Each of the methods has its specific features, advantages, and disadvantages. However, in the time analysis, a convolution operation, as a mathematical tool for analyzing the composition of processes, could be helpful.

1.1. Worst-case time analysis

The worst-case time analysis usually concerns industrial systems with strong time constraints (hard real-time system) [1]. Such systems require time determinism. The determinism is specified by values of the response time due to process requirements. From the perspective of the process, exceeding the maximum response time is unacceptable. Therefore, the primary parameter of evaluation of the system is the maximum response time. Time analysis is performed on the basis of pessimistic execution time of specific tasks in the system. These times are based on the maximum allowable time specification (time-out) for hardware and software and algorithms that determine the performance of a given system component.

This approach is sufficient to answer the question whether the system during the design phase meets time requirements imposed by the industrial process. However, it does not provide the information about a typical operation, in particular, about average processing time of the tasks. A statistical approach, which provides additional information on characteristics of a real working system, is taking into account time distributions of the tasks.

In the time analysis, the response times of the system are measured. As a result of a continuous observation of response times that occur at specific points of time, a discrete function of response times is obtained. By measuring the time for analysis, the area of analysis can be extended to analyze other values after recording the data containing the system response times. It can take into account not only the maximum values but also other values. The probability mass function of response times may be its representation. In this way the phenomena observed in the measurements can be interpreted statistically using the probability mass function.

However, the analysis and statistical interpretation of measurement results of the computer system response time turns out to be far more complex than the time analysis based on the expected time limits that result from time determinism.

1.2. Performance analysis

The performance analysis is a good method for finding errors in the process of creating and running the software, especially for systems operating in a continuous mode. The performance analysis usually involves the measurement of average values, whilst the time analysis is based on the analysis of maximum values. The following applications of the performance analysis of computer systems [2] can be distinguished:

- comparison of alternative solutions;
- checking the influence of new functionality on the system;
- tuning the system;
- monitoring the relative changes in system performance (acceleration, deceleration);
- detection of errors in creation and development of software;
- performance planning for solutions that do not exist yet.

Monitoring and the performance analysis is a fundamental detection test used by departments of quality control in the companies producing systems operating in a continuous mode. An important advance in these applications would be a possibility of transforming the methods of performance measuring of computer systems from a test of simple detection to the methods of diagnosis. The diagnostic tests can detect performance degradation. However, it is important not only to detect the degradation, but also to identify the component (subsystem), which caused the performance degradation. Diagnostic methods are intended to identify this element, such as locating the bottleneck in the system. Note also, that without the measurement process, it is not possible to enter any phase of validation or verification process in the performance evaluation study, presented in [3].

The performance analysis and time analysis are distinct areas of computing research. In the performance analysis the essential importance is attached to the probabilistic analysis and statistical models. In the time analysis of real-time systems the main problem is to identify time determinism in operation of the system. An open question is whether these areas have a common part. The method proposed in this article can be a common part of these two areas, with particular emphasis on the possibility of using measurements to diagnose the system. It may be an extension of measurement methods towards a diagnosis, in terms of determining the cause of changes in performance and searching for items that should be corrected.

1.3. Benchmarks

As mentioned before, in the time analysis of computer systems the worst-case analysis and analysis based on average values are normally used. Benchmarks are widely applied for comparing various properties of systems. On the other hand, the performance analysis usually measures average values. Such measurements are insufficient to identify many important characteristics of the system, for example the worst-case analysis, and do not provide much information which measurement data should provide. If there is degradation of performance, then benchmarks do not provide the answer to the question about the reason.

Based on the assumption that correct implementation works better and more efficiently than

implementation with functional errors, the prototype of the new system can be quickly diagnosed. Using the performance tests [4] for detecting errors and verifying the correctness of implementation requires:

- development of criterion for quantitative evaluation of system performance;
- development of measurement methods and measurements on referencing systems;
- development of model of the system;
- determination of asymptotic limits;
- searching for states of system performance and methods of identification;
- searching for measures of performance, based on parameters easily accessible by measurement;
- studying the statistical features of real systems;
- development of measurement methods for non-stationary systems.

One of the common approaches [5] applicable to the testing and debugging of prototype applications is measuring the performance [6]. Instead of decomposition of the system and verification of the particular components, subsystems, or functions, the performance of the whole system is examined.

2. Benchmark-type tests

Benchmarks usually measure performance for a particular workload. Let us analyze how the system behaves for such particular load. Symbols in the description of benchmarks presented here are used in the descriptions of queuing systems [7]. Let systems x, y, z have the same structure and consist of the following resources: processor *cpu*, disk *hdd* and network card *net*. Suppose that for the systems tests, workload references η_1, η_2, η_3 have been developed for the maximum workload of the individual elements of the system, i. e. *cpu*, *hdd* and *net* respectively. The measurement results indicate a measure of performance, which is assigned to the system by a benchmark test. During benchmark testing, when testing a single element of the system, other elements are also involved. For example, during the test of the drive *hdd*, the CPU, DMA and memory operations also take part.

Workloads η_1, η_2 and η_3 have the following properties. For fractional ϵ we choose the workload η_1, for maximum use of the resource *cpu*

$$\eta_1 \to U^{(cpu)} = 1 - \epsilon \tag{1}$$

where $U^{(cpu)}$ is resource utilization. Level of resource utilization is in the range [0-1]. Then the following relationship holds between the response time R of the system and the response time $R^{(cpu)}$ of the *cpu*.

$$R \geq R^{(cpu)} \tag{2}$$

In the test η_1 the resource with the maximum workload is the *cpu*. The resource *cpu* is the bottleneck then, so the system cannot have a better response time R than the response time of the most loaded resource. So similarly for the test η_2, where the most loaded resource is *hdd*.

$$\eta_2 \to U^{(hdd)} = 1 - \epsilon \tag{3}$$

then

$$R \geq R^{(hdd)} \tag{4}$$

For η_3 test

$$\eta_3 \to U^{(net)} = 1 - \epsilon \tag{5}$$

and then

$$R \geq R^{(net)} \tag{6}$$

Using a model of the system in which tasks performed by the system are served by resources, the average system response time $\bar{R}$, according to the Little law [8], [9], [10], [11], is

$$\bar{R} = \frac{\bar{N}}{X} - \bar{Z} \tag{7}$$

where $\bar{N}$ is the average number of clients in the system, X is the throughput, that is the number of tasks performed per time unit, Z is the average time (interval) between tasks generated for the system. Throughput is understood as defined in queuing theory and is the ratio of processed tasks during the observation time T. Using operational research approach we can determine the current value of X by counting the number of processed tasks in the system during the observation time T. Assuming that tasks are executed in series (not in parallel) and $D^{(i)}$ is working time of the

resource i, working time D of all resources is

$$D = \sum_i (D^{(i)}) \tag{8}$$

It is typically assumed for the benchmark tests that realization runs for one client only and there is no interval between tasks. So $Z = 0$ and $N = 1$. Thus, if there is only one task, this task does not wait, so the time of execution of the task in all resources is the response time of the system. So, if $N = 1$ then $R = D$. For each of the systems x, y and z the same rule applies, so

$$N = 1 \rightarrow R^{(i)} = D^{(i)} = B^{(i)}V^{(i)} \tag{9}$$

where $B^{(i)}$ is the execution time of the task in the resource i, $V^{(i)}$ is the number of visits (execution) of the task in the resource i. The working time of the resource consists of several visits of the task. In benchmark tests the value $B^{(i)}$ is usually large because typical tasks are long. Thus the number of visits $V^{(i)}$ in the resource is reduced.

Assume, that for the workload η_1 and the same number of visits V_{η_1} in the systems x, y and z, the system x has the shortest working time of the *cpu* resource. Thus the system x has the best *cpu* unit.

$$\min(D_x^{(cpu)}, D_y^{(cpu)}, D_z^{(cpu)}) = D_x^{(cpu)} \tag{10}$$

Similarly, assume that for the workload η_2 and the same number of visits V_{η_2}

$$\min(D_x^{(hdd)}, D_y^{(hdd)}, D_z^{(hdd)}) = D_y^{(hdd)} \tag{11}$$

For the workload η_3 and the same number of visits V_{η_3}

$$\min(D_x^{(net)}, D_y^{(net)}, D_z^{(net)}) = D_z^{(net)} \tag{12}$$

Little law [12] and assumptions $Z = 0$ i $N = 1$, usually adopted for benchmarks, show that

$$\bar{R} = \frac{1}{X} \tag{13}$$

The following conclusions may be drawn from the above considerations. For the workload η_1 the system x has always the shortest response time R of all systems x, y and z.

$$R_x = \min(R_i), i \in \{x, y, z\} \tag{14}$$

The throughput X_x of the system x is

$$X_x = max(X_i), i \in \{x, y, z\} \tag{15}$$

For the workload η_2:

$$R_y = min(R_i), i \in \{x, y, z\} \text{ and } X_y = max(X_i), i \in \{x, y, z\} \tag{16}$$

Similarly, for the workload η_3:

$$R_z = min(R_i), i \in \{x, y, z\} \text{ and } X_z = max(X_i), i \in \{x, y, z\} \tag{17}$$

The system with the best *cpu* always is the best in the test η_1, in which *cpu* is the most loaded resource.

The obtained results of the benchmark allow to determine the arrangement and relationship between systems x, y and z. For various benchmarks the measurements as response time R and throughput X are obtained. If the benchmark is based on R, then the system is better if it receives lesser value of the test result. If the benchmark is based on X, then the system is better if a greater value follows from the test.

Though the results obtained in this analysis are simple, they show that benchmark tests reflect only simple dependencies occurring in the system. In order to stimulate the system in real-world conditions, for example in test η_j, the number of clients must be more than one. We can also expect that the interval between tasks will be non-zero. Then the relations between $V^{(i)}$ and $B^{(i)}$ are changed, so

$$N > 1, Z > 0, V_{x,y,z}^{(i)} \neq V_{x,y,z}^{(\eta)}, B_{x,y,z}^{(i)} \neq B_{x,y,z}^{(\eta)} \tag{18}$$

In such case the formula (9) does not apply and the formula (7) is valid instead of (13). The conclusion is that benchmark tests do not reflect the complexity of executing tasks in the real system. They also cannot be used as a measurement method in the validation and projection phase [3].

More information on the system operation can be achieved from the deeper analysis of response times of the processes. The response time of the process is the composition of the response times of its subprocesses. For such an analysis the operations of convolution and deconvolution

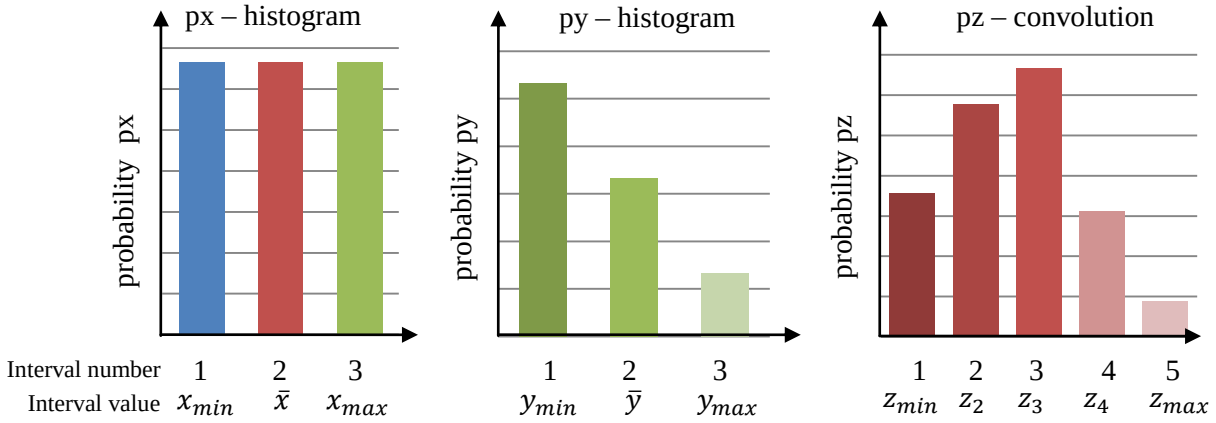

Figure 1. Histogram z calculated on base histograms of processes x and y for three ranges of values: minimum, mean, and maximum

performed on histograms of response times are helpful.

3. The convolution method in the time analysis of systems

Though convolution is well known method, its application in analyzing response time characteristics of particular components of complex system is a relatively novel approach [13], [14], [15], [16].

In the article it is particularly shown, that the convolution is relatively easy method to use, but the deconvolution is very sensitive when using for time characteristics measured in different times in independent components.

The presented method of time analysis of a compound computer system is based on convolution of two functions representing time-specific behavior of two subsystems that are parts of that system [17]. Convolution is considered one of the most important operations in the field of digital signal processing. Assume the existence of two independent subsystems x and y. A composition of both is a system z, so

$$z = x + y \tag{19}$$

The equation (19) represents also time dependences. It means that response time of system z is the sum of response times of subsystems x and y. Under this assumption it can be consequently stated, that the probability mass function pz of the response time of system z is equal to the convolution of probability mass function px and py of response times of subsystems x and y.

While executing the measurements for time analysis purposes, it is possible to record the average and minimum values, beside the maximum ones. The simplest case for introducing the method is when three ranges of values exist: minimum, mean, and maximum. The maximum value is collected during the worst-case analysis of a real-time system. The average and minimum values can be used in performance analysis. In addition to evaluation of the ranges, there is a need to know how often each value occurs. In the following discussion it is assumed that x and y are independent random variables, such as:

$$x \in \{x_{min}, \bar{x}, x_{max}\}; \quad y \in \{y_{min}, \bar{y}, y_{max}\} \tag{20}$$

For each value of the random variable, the probability of taking a given value can be determined. Thus, the probabilities that a variable represents the minimum, mean, and maximum value are known. The division of x and y into three values with the same intervals is used to derive the equations (21 - 23). The example of the convolution of two histograms is presented in Fig. 1. The consecutive probabilities of the resulting histogram pz for convolution shown in Fig. 1 could be calculated by simple formulas:

$$\begin{aligned} pz_{min} &= px_{min} py_{min}; \\ pz_2 &= px_{min} py_2 + py_{min} px_2 \end{aligned} \tag{21}$$

$$pz_3 = px_{min}py_{max} + px_2py_2 + px_{max}py_{min}; \quad (22)$$

$$pz_4 = px_2py_{max} + py_2px_{max}$$

$$pz_{max} = px_{max}py_{max} \quad (23)$$

The convolution method can be the useful tool for calculating time probability distributions. Thus, more information could be obtained than only minimum, average and maximum values. Referring to the definition of convolution, e.g., in [18], pz is a convolution of px and py.

$$pz = px * py = \int_0^\infty px(\tau)py(t-\tau)d\tau \quad (24)$$

The generalization of probability distribution pz for discrete values takes the following form:

$$pz_i = \sum_{j=0}^{i} px_j py_{i-j} \quad (25)$$

In the following considerations, the discrete counterparts of the continuous functions of probability distribution are used.

4. A case of compound process

Let us show an example of application of convolution for analyzing a simple process of writing data. Suppose the system uses the data buffering mechanism shown in Fig. 2. Buffering is implemented using two possible paths $x^{(1)}$ and $x^{(2)}$. The first one writes data to the buffer. This path is realized with probability p_1. The second path writes data to the buffer with t_0 delay, due to waiting for the access to the buffer. This path is realized with probability $1 - p_1$. These paths are modeled by two statistical processes with the exponential response time, shifted relative to each other at t_0 and probabilities: p_1 and $1 - p_1$ (Fig. 3a and Fig. 3b).

The two events are complementary. The distribution of the sum of these events (26) is shown in Fig. 3c. The write operation y is represented by the distribution py of response times, shown in Fig. 4b. It could be written as:

$$px = p_1px^{(1)} + (1 - p_1)px^{(2)} \quad (26)$$

where: $px^{(1)}$ – probability mass function of execution time $x^{(1)}$ of buffering with delay, $px^{(2)}$ – probability mass function of execution time $x^{(2)}$ of buffering without delay, px – probability mass function as result of compound execution time $x^{(1)}$ and $x^{(2)}$.

Thus

$$pz = px * py \quad (27)$$

where: py – probability mass function of time execution of write operation, pz – probability mass function of write operation with buffering.

The distribution of the sum of events (Fig. 4a) in convolution with the py distribution (Fig. 4b) gives the resulting distribution (Fig. 4c). The resulting distribution can be observed by measuring the system z response time for write operations. Such a specific characteristic (dual peek) can be detected in practice [19]. So, the observed distribution can be analyzed in a better way than while obtaining only minimum, average and maximum values. The characteristics is explained as a convolution of component processes [20].

Changing the time of waiting for the access to the buffer represented by parameter t_0 and changing the probability p_1 of this waiting, change the shape of the resulting distribution. Resulting distributions for $p_1 = 0.5$ and various t_0 are presented in Fig. 5.

Depending on the parameters of the model, different characteristics of system response time distributions can be obtained. The distributions can be in the form of one peek or even separated dual peeks in some cases. So the measured practical results can vary depending on the behavior of the process. Distributions for $t_0 = 55$ and various p_1 are presented in Fig. 6.

The advantage of the method based on convolution over other methods is the opportunity to observe the entire time probability distribution instead of selected values. Another advantage is the ability of simulation of behavior of the system for the case when some component or its time characteristic has to be changed. Then, by substituting the time distribution only for this component and then making the convolution with distributions of other components, the time distribution for the whole system can be calculated.

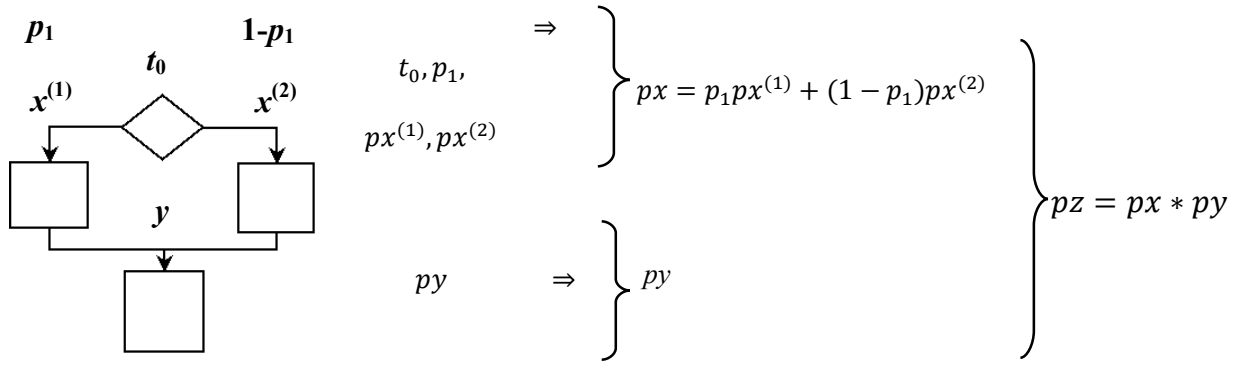

Figure 2. The example of system for writing with buffering

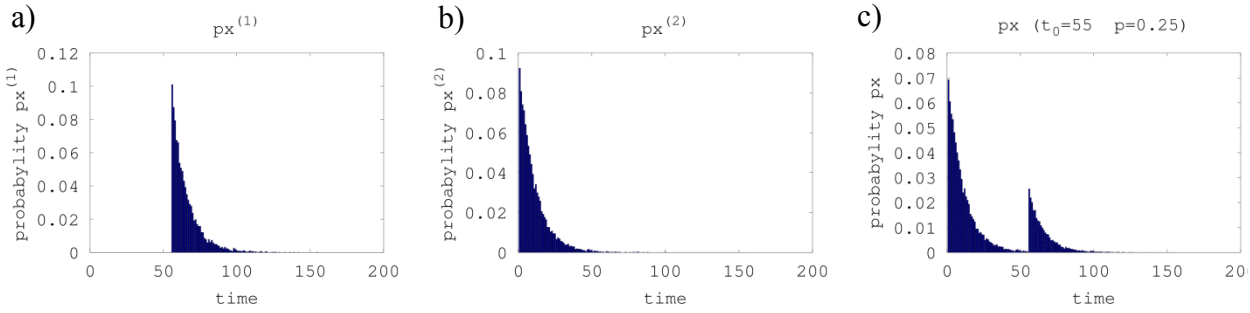

Figure 3. Distributions of response time for an example of a write system. $t_0 = 55$, $p_1 = 0.25$

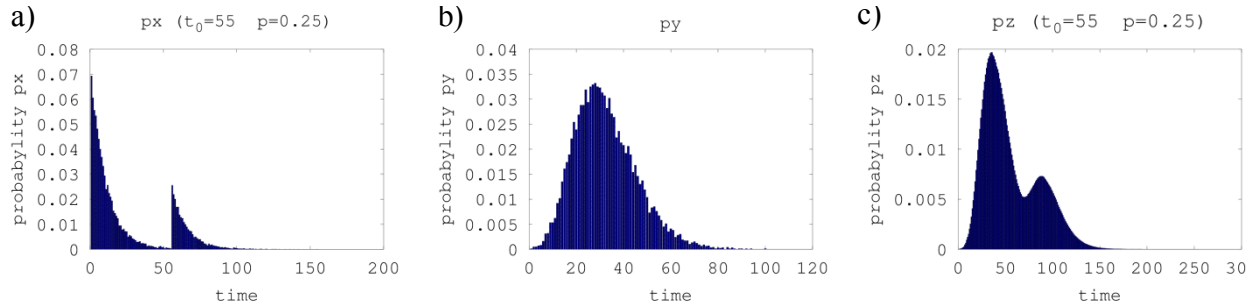

Figure 4. Resulting distribution pz as convolution of distribution px ('if' of events $x^{(1)}$ and $x^{(2)}$) with the distribution py

5. Analysis of measurement errors

In order to produce a good method of system analysis there is a necessity to consider that errors may occur in measurements. It may be caused by either inaccuracy of measurement or impossibility to measure all subsystems in the same time, so the measurements may be collected in different times. The result is the time inconsistency errors of consecutive measurement processes. Analysis of influence of measurement errors is important particularly in the operation of deconvolution. This process is not simple and clear because of sensitivity of deconvolution method. Convolution has the features as separation, commutativity, and associativity. The proofs are in [17].

The goal of deconvolution is to calculate time probability distribution for component y knowing time probability distributions of system z and

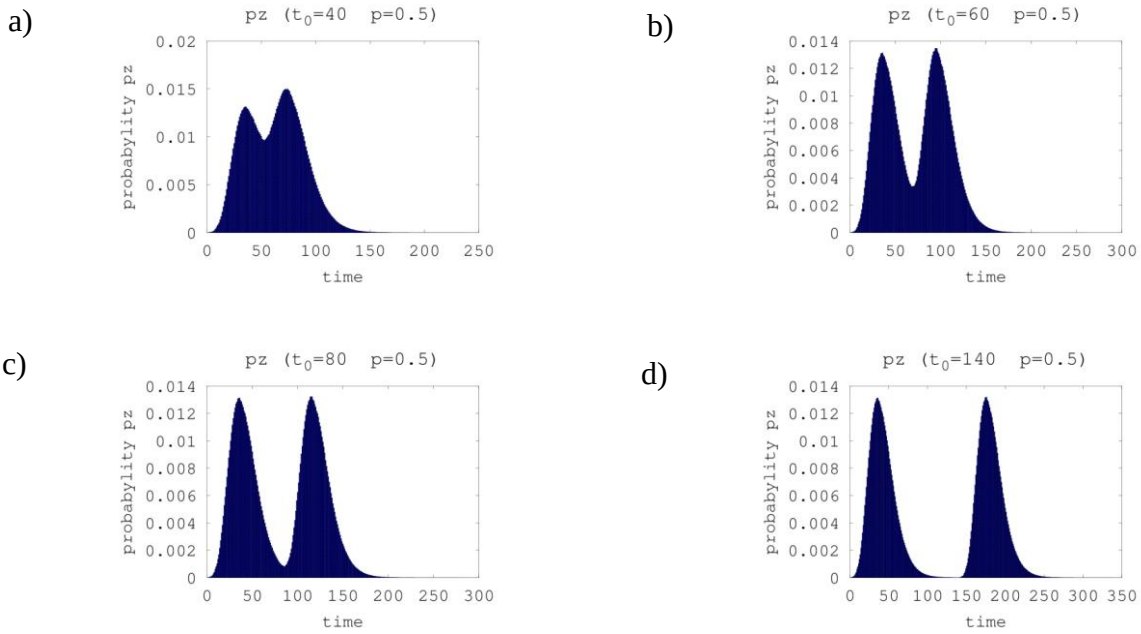

Figure 5. Distribution pz for $p_1 = 0,5$ a) $t_0 = 40$ b) $t_0 = 60$ c) $t_0 = 80$ d) $t_0 = 140$

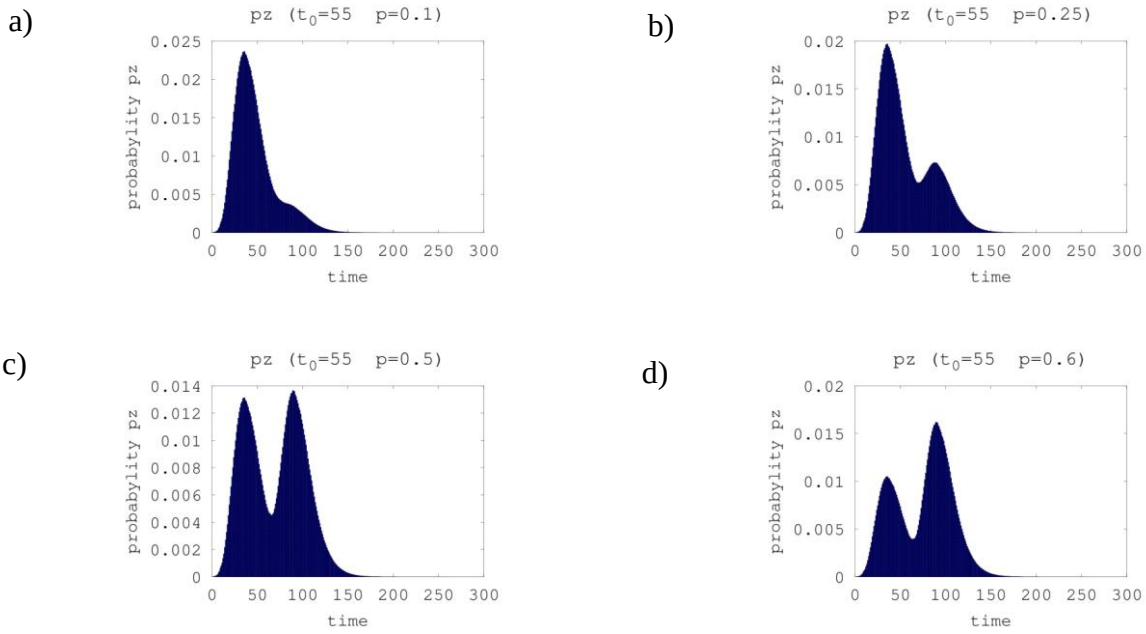

Figure 6. Distribution of pz for $t_0 = 0.5$ a) $p_1 = 0.1$ b) $p_1 = 0.25$ c) $p_1 = 0.5$ d) $p_1 = 0.6$

component x. If the form of an error in measurements z_m and x_m is known, then the result of equation (29) is identical to the solution of equation (28), under the condition that x and y are not distorted.

$$y = z - x \tag{28}$$

$$y = z_m - x_m \tag{29}$$

While measuring, a real response time, the z_m value is measured instead of z. It contains measurement errors e_x and e_y both for x and y, so

$$z_m = x + e_x + y + e_y \tag{30}$$

The total measurements error e consists of errors related to constituent processes (31).

$$e = e_x + e_y \tag{31}$$

Based on (30) and considering commutativity feature, the measured value z_m is:

$$z_m = x + y + e \tag{32}$$

To determine the value of y the two separated measurements have to be performed. During the one measurement the value x_m is collected (33):

$$x_m = x + e_x \tag{33}$$

During another measurement the value zm is collected (34):

$$z_m^{(error)} = x^{(error)} + y \tag{34}$$

There are errors acquired during the measurement of z_m. The $x^{(error)}$ is a response time from sub-system x during z_m measurement. These errors are immeasurable so instead of x_m the measured value is marked as $x^{(error)}$ in the case. It is also assumed that the values are similar (35).

$$x^{(error)} \approx x_m \tag{35}$$

The differences in the above values result from the facts that in the measurement of x_m the error e_y does not exist and also consecutive measurements of x are taken in different times. Thus, the assumption of the method while obtaining y is using x_m instead of $x^{(error)}$ in (34). From (34) the two mechanisms are considered. The first mechanism of occurring errors and the error analysis is done below in the form of (36).

$$y_{zm}^{(error)} = z_m^{(error)} - x \tag{36}$$

Formula (36) describes a situation in which the measurement error is introduced, because during the real system observation (z) the subsystem x behaves like x_m. Some environmental disturbances interact with the system x which do not exist during the observation of the isolated subsystem x. This disturbed subsystem is denoted as x_m. Unfortunately, the disturbances are not directly measurable. For the mean value the equations (37-40) represent the error introduced during the measurement.

$$\bar{y}_{zm}^{(error)} = \bar{z}_m^{(error)} - \bar{x} \tag{37}$$

$$\bar{y}_{zm}^{(error)} = \bar{x}_m + \bar{y} - \bar{x} \tag{38}$$

$$\bar{y}_{zm}^{(error)} = \bar{x} + e + \bar{y} - \bar{x} \tag{39}$$

$$\bar{y}_{zm}^{(error)} = \bar{y} + e \tag{40}$$

The result from (40) shows that measured mean value from y is charged with error e. If the calculations are not executed for the mean but for the probability mass functions p_{zm} and px, then the result is as below:

$$py_{zm}^{(error)} = pz_m^{(error)} \overset{/}{}_{deconv}\, px \tag{41}$$

$$py_{zm}^{(error)} = (px_m * py) \overset{/}{}_{deconv}\, px \tag{42}$$

$$py_{zm}^{(error)} = (px * pe * py) \overset{/}{}_{deconv}\, px \tag{43}$$

The equation (41) shows how the wanted py distribution can be obtained. The distribution found in such a way is laden by error pe convolved with the true distribution py (44):

$$py_{zm}^{(error)} = py * pe \tag{44}$$

The second mechanism of occurring errors during the measurements, other than shown in equation (37), is considered below:

$$y_{xm}^{(error)} = z - x_m \tag{45}$$

For the above formula (45) and during observation of the real system z the data is not affected by measurement errors derived from the subsystem x. Unfortunately, the interferences occur during the observation of the subsystem x and produce x_m. For the mean value the equations (46-48) represent error (45), introduced during the measurement.

$$\bar{y}_{xm}^{(error)} = \bar{z} - \bar{x}_m \tag{46}$$

$$\bar{y}_{xm}^{(error)} = \bar{x} + \bar{y} - \bar{x} - e \tag{47}$$

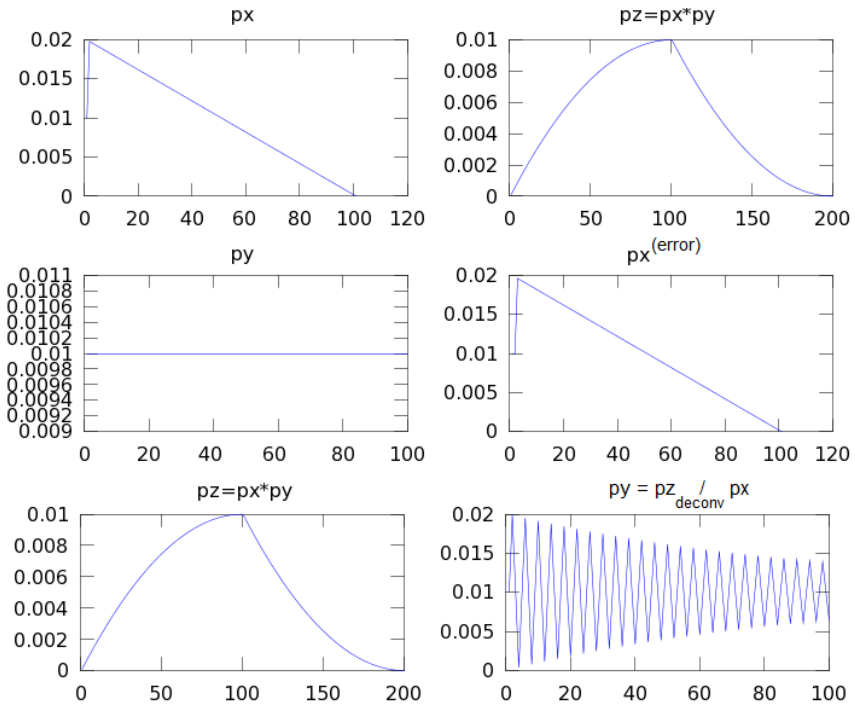

Figure 7. Example of the error influence: px, py, and pz on the left, pz, $px^{(error)}$ and deconvolved py on the right

$$\bar{y}_{xm}^{(error)} = \bar{y} - e \tag{48}$$

The equations (45) and (36) differ only in arithmetic operation in the equations (40) and (48). From the arithmetic point of view the mean value calculations (37-40, 46-48) are fully feasible for the any set of measurement data. Unfortunately, the situation changes radically for the application of deconvolution. The usage of inconsistent data i.e., collected from the consecutive measurement series taken at different times is not possible in a mathematically obvious way. The reason is high error sensitivity of the deconvolution operation. In order to perform the system decomposition (28), the data collected during the different ranges in the time of system activity can be used only with full awareness of the influence of the described error. Fig. 7 illustrates the phenomenon of deconvolution sensitivity. The example is based on the convolution of signals px and py and then the deconvolution of received pz and disturbed px marked as $px^{(error)}$. $px^{(error)}$ is slightly changed (near the value 2) in comparison to the original px. It can be seen, that small disturbance (error) of time probability distribution of component x causes very large disturbance in resulting probability distribution after deconvolution. For a probability mass functions pz and px_m, while using deconvolution, it is impossible to take into account the type of error (45), as shown in equations (49, 50).

$$py_{xm}^{(error)} = pz \overset{/}{_{deconv}} px_m \tag{49}$$

$$py_{xm}^{(error)} = (px * py) \overset{/}{_{deconv}} (px?+?pe) \tag{50}$$

Equation (50) by the usage of the operator '?+?' presents the mathematical problem which results in non-resistance of the methods of deconvolution to the error type (45). If pe is not convolved with px, then deconvolution made on the basis of the polynomial division method produces a large computational instability.

The presented considerations (37-50) show two different phenomena of errors. For the mean value the equations (37-40) compared to (46-48) differ only in the value of the error sign. Considering only the average values, the distinction of the mechanisms of error generation is unnecessary. However, it differs in the case of deconvolution of probability mass function. In the case (43-44) the considerations are fully mathematically correct.

The error values are added (convolved) (44) to the value of probability mass function $py_{zm}^{(error)}$. More precisely: added errors are the values of probability mass function pe. Unfortunately, using the deconvolution method is mathematically incorrect for the case (49-50). This is because the correctness of the physical implementation of measurement is not met. In the measuring method assumes that the measurements of the systems x and z are taken at different times. Thus the circumstances where the time signals are measured in physical conditions, in which the convolution of probability mass functions of signals exist, do not occur. It can be only supposed that for the separated measurements $z_1..z_l$ and $x_1..x_k$, and for the stable running conditions, the probability mass function px was the same in both measurements. Unfortunately, if px is changing during the measurement $z_1..z_l$ in relation to the measurement $x_1..x_k$, then the assumptions are not met and deconvolution method fails. In the calculation of the mean value (46-48) this problem does not occur.

6. Conclusions

While collecting measurement data, i.e., the response times, one does not receive only one state of the system, but a composition. Complex systems are non-linear, their analysis cannot be based on the simple method of measurement and analysis. The considerations conducted here show that measurements of the execution time of processes have much more complex statistical description than for example a simple description coming from benchmark tests. A statistical description delivers a dynamic character in contrary to a fixed value obtained in a given moment of system activity. The model of a queuing system, with appropriately selected assumptions, and ignoring insignificant details may, but may not, simulate basic rules of the system to be modeled.

The system usually works as many processes executed through many paths. The convolution method could be a useful tool for the analysis of behavior of complex processes and their time analysis, in a statistical meaning. Response times of the system, for many operations, sometimes give characteristic forms of probability mass functions that can be explained as the convolution of response times of subprocesses. Using the presented method, the entire time probability distributions of the system can be obtained. The method could also be used to simulate changes in any component of the system by simply modifying its probability mass function and then calculating the convolution. However, using modeling and convolution for diagnostics of a system requires experience. The analysis method depends heavily on the recorded measurement values. In such a modeling also measurement errors must be considered, particularly in deconvolution process. The deconvolution method may be also considered as a tool for such an analysis i.e., for obtaining time distribution of the component of the system for which the measurement cannot be collected. However, as shown above, deconvolution is a sensitive method and difficult for a practical application.

References

[1] G. Pratl, D. Dietrich, G. P. Hancke, and W. T. Penzhorn, "A new model for autonomous, networked control systems," *IEEE Transactions on Industrial Informatics*, Vol. 3, No. 1, feb. 2007, pp. 21 –32.

[2] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide.* Cambridge University Press, 2005. [Online]. http://books.google.pl/books?id=R8RLniX5DNQC

[3] E. D. Lazowska, *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, 1984. [Online]. http://books.google.pl/books?id=NNZQAAAAMAAJ

[4] D. A. Menascé, V. A. F. Almeida, L. W. Dowdy, and L. Dowdy, *Performance by design: computer capacity planning by example.* Prentice Hall PTR, New Jersey, USA, 2004.

[5] G. Bjedov, "Using open source tools for performance testing," *Google London Test Automation Conference (LTAC) Google Tech Talks*, September 2006. [Online]. http://video.google.com

[6] R. Blum, *Network performance open source toolkit: using Netperf, tcptrace, NIST Net, and SSFNet.* Wiley Pub., 2003. [Online]. http:

//books.google.pl/books?id=ECt5ycQ9D7YC
[7] T. Czachórski, *Modele kolejkowe systemów komputerowych*, ser. Skrypty Uczelniane - Politechnika Śląska. Wydawnictwo Politechniki Śląskiej, 1994. [Online]. http://books.google.pl/books?id=50KkPgAACAAJ
[8] J. D. C. Little, "A proof for the queuing formula: $\mathsf{L} = \lambda\mathsf{W}$," *Operations Research*, Vol. 9, No. 3, May/June 1961, pp. 383–387.
[9] S. Stidham, Jr., "A last word on $\mathsf{L} = \lambda\mathsf{W}$," *Operations Research*, Vol. 22, No. 2, March/April 1974, pp. 417–421.
[10] F. Beutler, "Mean sojourn times in Markov queueing networks: Little's formula revisited," *Information Theory, IEEE Transactions on*, Vol. 29, No. 2, mar 1983, pp. 233 – 241.
[11] P. W. Glynn and W. Whitt, "Extensions of the queueing relations $\mathsf{L} = \lambda\mathsf{W}$ and $\mathsf{H} = \lambda\mathsf{G}$," *Operations Research*, Vol. 37, No. 4, July/August 1989, pp. 634–644.
[12] L. Lipsky, *Queueing Theory A Linear Algebraic Approach*. Springer New York, 2009. [Online]. http://dx.doi.org/10.1007/978-0-387-49706-8_3
[13] J. P. Lehoczky, "Real-time queueing theory," in *Real-Time Systems Symposium, 1996., 17th IEEE*, dec 1996, pp. 186 –195.
[14] J. L. Diaz, D. F. Garcia, K. Kim, C.-G. Lee, L. Lo Bello, J. M. Lopez, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, 2002, pp. 289 – 300.
[15] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, Vol. 85, No. 5, 2012, pp. 1147 – 1156. [Online]. http://www.sciencedirect.com/science/article/pii/S0164121211003232
[16] M. Santos, B. Lisper, G. Lima, and V. Lima, "Sequential composition of execution time distributions by convolution," *Proc. 4th Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2011)*, November 2011, pp. 30–37.
[17] J. Yeh, *Real analysis: theory of measure and integration*. World Scientific, Danvers, USA, 2006. [Online]. http://dx.doi.org/10.1007/978-0-387-49706-8_3
[18] A. Mattuck, *Introduction to analysis*. Prentice Hall, 1999. [Online]. http://books.google.pl/books?id=N0FkQgAACAAJ
[19] S. Wideł, J. Flak, and P. Gaj, "Interpretation of dual peak time signal measured in network systems," in *Computer Networks*, ser. Communications in Computer and Information Science, A. Kwiecień, P. Gaj, and P. Stera, Eds. Springer Berlin Heidelberg, 2010, Vol. 79, pp. 141–152. [Online]. http://dx.doi.org/10.1007/978-3-642-13861-4_14
[20] ——, "Analysis of time measurements in network systems using decomposition on subprocesses," in *Computer Networks*, ser. Communications in Computer and Information Science, A. Kwiecień, P. Gaj, and P. Stera, Eds. Springer Berlin Heidelberg, 2011, Vol. 160, pp. 70–79. [Online]. http://dx.doi.org/10.1007/978-3-642-21771-5_9

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 65–76, DOI 10.5277/e-Inf130107

Static analysis of function calls in Erlang

Refining the static function call graph with dynamic call information by using data-flow analysis

Dániel Horpácsi*, Judit Kőszegi*

*Department of Programming Languages and Compilers, Eötvös Loránd University, Budapest, Hungary

daniel_h@inf.elte.hu, kojqaai@inf.elte.hu

Abstract

Functions and their relations can affect numerous properties and metrics of a functional program. To identify and represent the functions and their calling connections, software analysers commonly apply semantic function analysis, which derives the static call graph of the program, based on its source code. Function calls however may be dynamic and complex, making it difficult to statically identify the callee. Dynamic calls are determined just at run-time, static analysis therefore cannot be expected to fully identify every call.

Nevertheless, by utilising the results of a properly performed data-flow analysis as well as taking ambiguous references into account, numerous dynamic calls are discoverable and representable. We consider cases where the identifiers of the callee are statically determined, but they flow into the call expression from a different program point, and also, we manage to handle function calls whose identifiers are not fully identifiable at compile-time. By utilising the improved reference analysis, we extend the static call graph with various information about dynamic function calls. We investigate such a function call analysis in the programming language Erlang.

1. Introduction

To overview the components of the software, to identify relations and dependencies, and to find out properties, we can apply static source code analysis. The analysis can be followed by semi-automatic code transformations correcting design weaknesses, based on the analysis results. Both software transformation tools and reverse engineering techniques operate on programs, and involve many sorts of static code analysis.

Consider the case of refactoring tools [1, 2], where code transformations never should change the semantics of the program being refactored. Despite the fact that such automatic code transformation tools often involve user interaction, the code modifications made are directed mostly by the information gathered from the source code. Consequently, the correctness of the transformations highly depends on the accuracy of the static analysis carried out before the actual transformation steps.

In this paper we focus on the analysis of inter-procedural relationships. In different programming languages there are different call constructs, including dynamic ones that may be unidentifiable at compile-time and therefore usually are omitted by static analysers. However, the data bindings that determine dynamic calls may be looked up by use of static data-flow analysis, and if the function identifiers are statically given in the code, we can successfully locate them and identify the referred function. We concentrate on the analysis of call constructs present in Erlang [3, 4], a dynamically typed functional programming language. The presented approach aims to refine function call graphs [5] by means of static analysis of dynamic function calls. By utilising the more sound call graph we can improve different sorts of static

analysis as well as refactoring code transformations.

In the next sections we introduce the main types of function call constructs – including dynamic ones – and also we precisely define the connection between call expressions and function entities in terms of formal semantic rules. In addition, concepts of ambiguous dynamic calls and opaque functions are introduced in order to represent partially unidentifiable function references as well. Finally, we define formal relationships between the function entities, merging all the call information into the call graph and show a simple case study.

2. Call constructs

We suppose that the functions of a program are grouped into modules and are dynamically typed (like in Erlang). With this, a function may be identified by a 3-tuple (so-called *MFA*) that includes the name of the module the function is located in, the name of the function, and the number of its formal parameters (called 'function arity'). This function descriptor can be written in the form of *module:function/arity*.

	MFA-call	Apply-call
identifiers as literals	static	dynamic
identifiers as expressions	dynamic	dynamic

Figure 1. Static and dynamic calls

Call constructs are sorted in order to ease their analysis; Figure 1 shows the main groups. Syntactically, we consider two types of function call: MFA-calls and apply-calls. The former one is the common way to invoke a subroutine, while apply-calls may be regarded as symbolic calls so that they refer to a special function named 'apply' which then results in another call. For this use, both the function to be called and its arguments are specified within the arguments of the apply-call. When called, it executes the named function on the specified arguments and then returns its result. Basically, apply-calls behave very similarly to MFA-calls, however, the main difference in use lies in the way the parameters are constructed and then passed to the callee.

Beside the syntactic grouping, we make a distinction between static and dynamic call methods. Function calls may affect the data-flow within the program, and interestingly, data-flow may also take effect on function calls so that function calls may be constructed by means of run-time data. Programming languages usually support meta-programming techniques like handling the program itself as data or creating statements at run-time ('eval' methods). A special case of the latter technique is the run-time construction of function calls, where the identifier(s) of the called routine may be determined just at execution-time, similarly to the construction of actual function parameters. There are many programming languages that support meta-programming (and dynamic call constructs), including script languages (like JavaScript, Ruby, Python) as well as functional languages, such as Erlang and Scheme.

2.1. MFA-calls

In Erlang, MFA function calls provide the standard way of executing a function from inside another one. These call expressions can be written using the following syntax:

$$module\_name{:}function\_name(arg1, ..., argN)$$

Each of *module_name* and *function_name* must be either an identifier (atom literal) or an Erlang expression that evaluates to an identifier determining the name of the callee. Observe that the arity is fully defined at compile-time, with the number of parameters actually enumerated between the parentheses. When a call is written in the above syntax and both function identifiers are given as atom literals, we say that the function call is a static MFA-call. It is said to be static because the identifiers of the function are given statically, at the location of the call. Most static analyser software can successfully observe such call constructs and can build the corresponding call graph, however, they simply omit dynamic call methods.

In dynamic MFA-calls, either the module name or the function name is given with non-literal expressions (for instance, with variables). Static analysis of such calls requires some kind of data-flow analysis which uncovers the origin of the data (for example, the values bound to the variables).

2.2. Apply-calls

As we already mentioned, there is another call construct in Erlang, the so-called apply-call, which is based on a built-in higher-order function called `apply`. (We note that similar call constructs exist in various programming languages.) This construct is dynamic by nature, since the identifiers of the called function are given by means of the arguments of another routine, which is evaluated certainly just at run-time. The apply-calling expressions can be written using the following syntax:

apply(*module_name*,*function_name*,*arg_list*)
where $arg\_list \equiv [arg1, ..., argN]$

The call primarily refers to a built-in function called `apply` that is located in the module `erlang` (one may refer to it as *erlang:apply/3*). Its parameters determine the secondarily referred function that is being called at run-time. Each of *module_name* and *function_name* must be either an identifier (atom literal) or an Erlang expression that evaluates to an identifier, while *arg_list* should evaluate to an Erlang list whose length precisely determines the arity of the callee at run-time.

Listing 1 shows a simple apply-call. Since the identifiers are given as atom literals, one may regard this simple case as a static apply-call. However, as we already stated, apply-calls are dynamic by their nature, so in this paper we will treat and analyse apply-calls as fully dynamic constructs. In Listing 2 a more complex example shows a function call referring to the same function whilst demonstrating possible difficulties of static analysis implemented on dynamic function calls. Note that we already came to the need of data-flow analysis and additional static analysis methods, which motivates us to examine our possibilities on static analysis of dynamic function calls.

Listing 1. Simple apply-call

```
apply(io, format, ["hello", []]).
```

Listing 2. More complex apply-call

```
f() -> {format, io}.
g() -> {F, M} = f(),
       Rest = [[]],
       Args = ["hello" | Rest],
       apply(M, F, Args).
```

3. Static analysis of function calls

In the context of static, context-insensitive call analysis, "function" stands for an identifier (or a signature) of a routine, that is a (possibly minimal) set of data that can unambiguously identify the routine. Basically, static function analysis aims to extract these function descriptors and their calling connections into a sound function call graph, which can be used within further analysis or program transformations. We assume that each routine of the program under analysis has such a function descriptor (also referred to as semantic function entity).

In the following sections we define the connection between expressions of the source code and semantic function entities involved in the program, in terms of formal semantic rules. While describing the semantics, we suppose that the program code is represented by an abstract syntax tree, so semantic rules instance syntax elements as subtrees of the semantic program graph (3-layered, labelled extension of the abstract syntax tree, including the static semantics of the program). Most of the concepts described in the paper have been implemented in RefactorErl [2], a source code analyser and transformer tool, wherein many other kinds of static semantic analysis can be carried out on Erlang programs [6].

Data-flow analysis. The analyser framework of RefactorErl includes, in addition to many kind of analysis, a data-flow analyser, which is able to carry out 0^{th} order and 1^{st} order data-flow anal-

ysis [7]. Its backward data-flow reaching relation returns all the expressions affecting the queried one. However, in the case of dynamic call analysis we only need the ends of the reaching paths, that is, those nodes that may potentially uncover the possible values of expressions. We introduce the concept of compact data-flow reaching, which performs pure closure on the data-flow relation and consequently returns only the nodes in which the reaching terminates. If such an expression is a literal, we found a possible value of the expression. Within the dynamic call analysis, we use the 0^{th} order compact backward data-flow reaching relation.

Auxiliary definitions. Previously we shortly introduced the syntax of Erlang function call expressions. In this paper we only use a small subset of the language syntax, on which we build in the course of defining semantic rules, so the syntax of the whole language is not specified by formal means. Basically, Erlang programs consist of forms (mostly functions) grouped into modules, where each function embodies a sequence of expressions. In Erlang programs, atoms (named constants) are used to identify entities, e.g. modules and functions.

In the rest of the paper,

- E_{Atom} denotes the set of Erlang atom literal expressions
- E_{List} is the set of all expressions that construct list values
- E denotes the set of all Erlang expressions (including E_{Atom} and E_{List} as well).

We define some sets of semantic values (domains):

- $Atom$ = set of possible atom values
- $Atom' = Atom \cup \{\bot\}$
- $\mathbb{N}^{\geq} = \{n^{\geq} | n \in \mathbb{N}\}$ where $n^{\geq} \equiv [n..\infty)$
- $\mathbb{N}' = \mathbb{N} \cup \mathbb{N}^{\geq} \cup \{\bot\}$

And also, we define a total ordering on the elements of $\mathbb{N}^{\geq}$ $(n, m \in \mathbb{N})$:

$$n^{\geq} \leq m^{\geq} \qquad \text{if } n \leq m$$

Now, the following functions are defined over the syntactic elements and map onto the semantic domains, giving the bridge between syntax and semantics.

$Val : E_{\boldsymbol{Atom}} \mapsto Atom$
: $Val(e)$ returns the value given by the evaluation of the atom expression e.

$Length : E_{\boldsymbol{List}} \mapsto \mathbb{N}'$
: $Length(e)$ gives the length of the Erlang list value represented by the expression e. Note that it maps to $\mathbb{N}'$ and therefore may return either a concrete number, a lower bound, or the $\bot$ symbol.
If the list length is only partially analysable and thus a lower bound is calculable, then $Length(e) \in \mathbb{N}^{\geq}$. Also, if we cannot calculate the list length at all, then $Length(e) = \bot$.

$\overset{0f_{cb}}{\rightsquigarrow} \subseteq E \times E$
: $e_1 \overset{0f_{cb}}{\rightsquigarrow} e_2$ means that the value of e_1 flows into e_2 in 0^{th} order using compact reaching [7].

Finally, we define the set of semantic function entities and define a function that returns such entities based on their 3-tuple descriptor. Note that each of the function identifiers may be undefined ($\bot$).

$SemFun$
: The semantic function entities involved in the analysed program

$Function : Atom' \times Atom' \times \mathbb{N}' \mapsto SemFun$
: $Function(m, f, a)$ returns the function entity identified by its module name, function name and arity.

In the following, $e_1 \xrightarrow{L} e_2$ denotes a binary relation between e_1 and e_2, which is a directed graph edge (being labelled by L) between the two graph nodes (e.g. expression occurrences) on the implementation level.

3.1. Semantics of MFA-calls

In order to define the syntax of MFA-calls, we use the previously introduced sets of language elements. The abstract syntax of an MFA-call is the following.

$$e_{MFA} \equiv e_{MN}{:}e_{FN}(e_1, \ldots, e_n)$$

In the above line, e_{MFA} is a node belonging to an MFA-call expression referring to a function with exactly n parameters. We only assume that the

module name (e_{MN}), the function name (e_{FN}), and the actual parameters ($e_1, \ldots, e_n$) are given as Erlang expressions.

$$e_{MFA}, e_{MN}, e_{FN}, e_1, \ldots, e_n \in E$$

The following semantic rules define relations between syntactic elements and semantic entities. We have already seen that the parameters of an MFA-call are explicitly enumerated within the call, thus the arity of the callee is easy to calculate. Consequently, the potential difficulties may arise during the analysis of module and function names, since they may flow into the expressions e_{MN} and e_{FN} from an arbitrarily far point of the program (e.g. from another module or another application). Our goal is to uncover the possible values of these expressions by use of data-flow analysis in order to refine the call graph with the dynamic call relations.

Static calls. First of all, we define the rule of static MFA-calls, shown in rule MFA1. It is pretty straightforward, but apparently a necessary part of the model. In this case both the module name and the function name are given as atom literals, so the identifiers are given just at the point of the call. One can see that the values of the atom expressions together with the parameter count exactly identify the function being referred. The call expression is linked to the function entity labelled by *funref* (static function reference).

Fully identifiable dynamic calls. When the module name or the function name is not explicitly given as an atom literal, however, by applying data-flow reaching we can successfully find out some possible values of the expression(s), we can identify some possible callee. Such references are said to be dynamic and unambiguous.

The call expression is linked to all the possible functions, labelled by *dynref* (dynamic function reference). Listing 3 shows a dynamic MFA-call in which the identification of the module name requires data-flow analysis. The name comes from another function call, the outer call is analysed by using the rule MFA2.

Listing 3. Fully identifiable dynamic MFA-call

```
iomodule() -> io.
f() -> (iomodule()):format("hello",[]).
```

Ambiguous function calls. Now let us define a previously not detailed kind of call reference. Namely, in the case if an element of the function descriptor cannot be determined by using data-flow analysis either, we are not able to fully identify the potentially referred functions. The reason why we are not able to calculate, for example, a function name, is that the data-flow path ends not in an atom literal but in another kind of expression from which the reaching cannot be continued. In order to be able to consider such cases, we introduce the concept of ambiguous function references, where one identifier of the callee is unable to be precisely calculated. Listing 4 demonstrates a call whose module reference is unknown. Even in this case we note a function reference, however, not to a fully defined function. Instead, we define opaque functions and create references to these special function entities.

We note that we do not deal with function calls not specifying at least two of the three main identifiers of a function (module name, function name, and arity). Opaque functions consequently only have exactly one undefined field in their descriptor. In addition, there is a special case that may appear during the analysis of apply calls, namely, when the argument list is only partially present, and based on it we can calculate a lower bound of the function arity. This issue will be detailed in the section of apply-call analysis. An overview of dynamic/ambiguous calls and their analysis is present in Figure 2.

Partially identifiable MFA-calls. As we mentioned, in the case of ambiguous references one of the three main function identifiers is unable to be determined by means of static analysis. Since the arity is exactly given by syntax of MFA-calls, the uncertainty may come from the identification of the names.

Suppose that the function name is determinable. If there is a case in which we cannot determine the name of the referred module, we cannot completely identify the potentially referred functions. In order to indicate this, we create a reference that points to an opaque function whose module name is unknown ($\perp$). See rule MFA3.

If the module name is determinable and the function name is not, we analogously get to the rule for MFA-calls with unidentifiable function names (see rule MFA4).

Listing 4 demonstrates a function call where the function name comes from a `case` expression and can be either "foo" or an arbitrary atom read from the standard input. Observe that while analysing this example we should apply each of rule MFA2 and rule MFA4, since the first case clause gives a fully identified name, in contrast with the second one, which refers to a value that is unknown at compile-time. Consequently, the call expression is related to two different functions: with *dynref* to a fully defined function, and with *ambref* to an opaque function entity.

Listing 4. Ambiguous MFA-call

```
Fun = case read_int() == 0 of
        false -> foo;
        true  -> read_atom()
      end,
module:Fun(ok, 0)
```

In Listing 5 we show a call that is skipped during the function analysis in order to avoid storing calling relations that are not specified enough and would excessively expand the call graph.

Listing 5. Unanalysed MFA-call

```
{Mod, Fun} = {read_atom(), read_atom()},
Mod:Fun(0)
```

3.2. Semantics of apply-calls

An apply-call may be regarded as a meta-call that primarily refers to the *erlang:apply/3* built-in function and secondly refers to the function specified in the call parameters. The abstract syntax of apply-calls is the following.

$$e_{APP} \equiv \mathsf{apply}(e_{MN},e_{FN},e_{Args})$$

In the above line, e_{APP} is a node belonging to an apply-call expression. We only assume that the module name (e_{MN}), the function name (e_{FN}), and the actual list of parameters (e_{Args}) are given as legal Erlang expressions (thus the argument list does not have to be a list expression actually).

$$e_{APP}, e_{MN}, e_{FN}, e_{Args} \in E$$

The following semantic rules define dynamic call relations between call expressions and function entities.

Fully identifiable apply-calls. In case of apply-calls, the module and function names as well as the argument lists may be constructed arbitrarily far from the call point. Provided that applying data-flow reaching we can successfully find out the possible value of the name expression(s) as well as the length of the argument list, we can fully identify the callee. Such cases are called to be dynamic, unambiguous references. The call expression is linked to all the possibly referred functions, labelled by *dynref*. Listing 6 shows an apply-call that requires data-flow analysis, but it is still possible identify the callee by using rule APP1.

Listing 6. Fully identified apply-call

```
MN   = io,
FN   = format,
Args = ["hello"],
apply(MN, FN, Args)
```

Partially identifiable apply-calls. Similarly to MFA-calls, apply-calls may be ambiguous, which means we cannot certainly identify every callee. Any component of the 3-tuple identifying

	MFA-call	*Apply-call*
all identifiers are calculable	dynamic	dynamic
one of the identifiers is incalculable	ambiguous	ambiguous
module and function names plus a lower bound of the arity are calculable	—	ambiguous
at least two key identifiers are incalculable	skipped	skipped

Figure 2. Dynamic call types in detail

$$\frac{}{e_{MFA} \xrightarrow{funref} Function(Val(e_{MN}), Val(e_{FN}), n)} \; e_{MN} \in E_{Atom},\ e_{FN} \in E_{Atom} \tag{MFA1}$$

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_y \overset{0f_{cb}}{\rightsquigarrow} e_{FN}}{e_{MFA} \xrightarrow{dynref} Function(Val(e_x), Val(e_y), n)} \; e_{MN} \notin E_{Atom} \vee e_{FN} \notin E_{Atom},\ e_x \in E_{Atom},\ e_y \in E_{Atom} \tag{MFA2}$$

$$\frac{e_z \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_x \overset{0f_{cb}}{\rightsquigarrow} e_{FN}}{e_{MFA} \xrightarrow{ambref} Function(\bot, Val(e_x), n)} \; e_x \in E_{Atom},\ e_z \in E \setminus E_{Atom} \tag{MFA3}$$

the function may be incalculable at compile-time, resulting in uncertain function references. Interestingly, due to the way the arguments are passed to the function, there may appear situations where only a lower bound of the function arity is calculable.

Suppose that the function name along with the arity are determinable. If there is a case in which the name of the referred module cannot be calculated by use of data-flow analysis either, we cannot fully identify the potentially referred functions. In order to indicate this, the call expression is linked to an opaque function entity whose module name is undefined (see rule APP2). The rest of the identifiers are read out from the code, while the relation is labelled by *ambref* (ambiguous function reference).

We analogously construct a rule for ambiguous apply-calls with an incalculable function name (see rule APP3). The expression e_{APP} is linked to an opaque function whose containing module name and arity can be read out from the code, however, its name is set to $\bot$ (undefined). The reference is labelled by *ambref*.

Listing 7. Ambiguous apply-call

```
MN = io,
FN = if read_int() == 1 -> format;
        true            -> read_stdin()
     end,
apply(MN, FN, ["hello"])
```

Listing 7 shows an example in which the function name comes from a conditional statement. The analysis uncovers that it may have the value `format`, but on the other hand, it may come from the standard input as well. This results in two relations, based on the rules APP1 and APP3: a dynamic reference goes to *io:format/1* and an ambiguous one to *io:*$\bot$/1.

Now suppose that the module name and the function name are determinable. If we cannot gather any information about the arity of the function by use of data-flow analysis either, a reference points to an opaque function whose arity component is unknown ($\bot$), indicating that the function reference is ambiguous in the arity (see rule APP4).

Sometimes, even if we cannot determine the arity, we still have a chance to calculate a lower bound of the parameter count (it happens if the length of the tail of an argument list is incalculable). If we can uncover such a bound, it will be indicated beside the fact that the function reference is ambiguous. To avoid creating lots of opaque functions, we do not associate separate functions to each different lower bound of the arity the call may refer to. Instead, we identify the greatest lower bound and we link only one opaque function to the call, using the minimum of the lower bounds as arity. Let us define the following predicate.

$$AL(e_{List}) = e_{List} \overset{0f_{cb}}{\rightsquigarrow} e_{Args} \wedge Length(e_{List}) \in \mathbb{N}^{\geq}$$

So $AL(e)$ is true if and only if e flows into the argument list of the call and its length is not fully known, but its lower bound can be calculated with static analysis.

By utilising the AL predicate, we can give the rule for the calls with lower-bounded arities (see APP5). The rule shows that even if many lower-bounds of the arity are calculable, we unify them into a single one, which is actually the

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_z \overset{0f_{cb}}{\rightsquigarrow} e_{FN}}{e_{MFA} \xrightarrow{ambref} Function(Val(e_x), \bot, n)}\ e_x \in E_{Atom},\ e_z \in E \setminus E_{Atom} \tag{MFA4}$$

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_y \overset{0f_{cb}}{\rightsquigarrow} e_{FN} \qquad e_L \overset{f0_{cb}}{\rightsquigarrow} e_{Args}}{e_{APP} \xrightarrow{dynref} Function(Val(e_x), Val(e_y), Length(e_L))}\ e_x, e_y \in E_{Atom},\ e_L \in E_{List},\ Length(e_L) \in \mathbb{N} \tag{APP1}$$

$$\frac{e_z \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_x \overset{0f_{cb}}{\rightsquigarrow} e_{FN} \qquad e_L \overset{f0_{cb}}{\rightsquigarrow} e_{Args}}{e_{APP} \xrightarrow{ambref} Function(\bot, Val(e_x), Length(e_L))}\ e_x \in E_{Atom},\ e_z \in E \setminus E_{Atom},\ e_L \in E_{List},\ Length(e_L) \in \mathbb{N} \tag{APP2}$$

greatest lower bound of the arity. As $\mathbb{N}^{\geq}$ is a totally ordered set, the function *minimum* can be used to get the minimal element.

Listing 8. Lower bounded arity in an apply-call

```
msg() -> if is_young(user()) ->
             [$h,$i,$ |read_stdin()];
           true ->
             [$h,$e,$l,$l,$o,$ |read_stdin()]
         end.
f() -> apply(io, format, msg()).
```

In listing 8 we demonstrate the use of rule APP5. In this example there are two different lower-bounded argument lists belonging to the call: one with length $3^{\geq}$, and another one having $6^{\geq}$ elements. Consequently, the greatest lower bound is 3, and thus the arity of the ambiguously referred opaque function is $3^{\geq}$.

The "may be" relation

Let us define

$$mfa : SemFun \mapsto Atom' \times Atom' \times Int'$$

where $mfa(f)$ results in the 3-tuple function descriptor identifying f. We claim that

$$mfa(Function(m, f, a)) = (m, f, a)$$

In the previous sections we have introduced dynamic and ambiguous function references, which relate expressions to function entities. Also, we presented the use of opaque functions, giving the opportunity to precisely represent ambiguous references. However, these opaque functions are special, they may not be regarded as legal elements of the function call graph.

To integrate opaque functions into the call graph, the first step we do is associating these functions to fully defined ones. This relation is called *may_be*, and it connects opaque functions to non-opaque ones that are potential targets of ambiguous calls. Namely, if the unambiguous and the ambiguous functions are identical in the two identifier components defined in the opaque function, the concrete function corresponds to the opaque one. In other words, the concrete function may only differ in the one identifier that is undefined in the opaque function. This relation is defined by rule MAY1.

A special kind of opaque function has a lower bound of its arity. While looking for possible *may_be* connections, one has to take into account not only the names but also the lower bound of the opaque function. A concrete function entity may be associated with a fully defined one only if their names are equal and the concrete arity complies with the lower bound (see rule MAY2).

With the above relation we successfully included the information about ambiguous references and opaque functions into the model of semantic functions by associating opaque entities with concrete ones. Thus we do not have to consider and directly handle opaque functions during further analyses.

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_z \overset{0f_{cb}}{\rightsquigarrow} e_{FN} \qquad e_L \overset{f0_{cb}}{\rightsquigarrow} e_{Args}}{e_{APP} \overset{ambref}{\longrightarrow} Function(Val(e_x), \bot, Length(e_L))}\ e_x \in E_{Atom},\ e_z \in E \setminus E_{Atom},\ e_L \in E_{List},\ Length(e_L) \in \mathbb{N} \tag{APP3}$$

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_y \overset{0f_{cb}}{\rightsquigarrow} e_{FN} \qquad e_L \overset{f0_{cb}}{\rightsquigarrow} e_{Args}}{e_{APP} \overset{ambref}{\longrightarrow} Function(Val(e_x), Val(e_y), \bot))}\ e_x \in E_{Atom},\ e_y \in E_{Atom},\ e_L \in E_{List},\ Length(e_L) = \bot \tag{APP4}$$

$$\frac{e_x \overset{0f_{cb}}{\rightsquigarrow} e_{MN} \qquad e_y \overset{0f_{cb}}{\rightsquigarrow} e_{FN} \qquad e_{List} \overset{f0_{cb}}{\rightsquigarrow} e_{Args}}{e_{APP} \overset{ambref}{\longrightarrow} Function(Val(e_x), Val(e_y), Arity)}\ e_x, e_y \in E_{Atom},\ e_{List} \in E_{List},\ Length(e_{List}) \in \mathbb{N}^{\geq} \tag{APP5}$$

$$\text{where} \quad Arity = min\{Length(e_{List}) \mid e_{List} \in E_{List} \wedge AL(e_{List})\}$$

Extending the call graph

Let us define the function

$$Body : SemFun \mapsto \mathcal{P}(E)$$

where $Body(f)$ contains all expressions that are inside the function body of f (if f is actually defined in the code). If f has no definition, then $Body(f)$ is an empty set.

Consider a function call expression e and the function entity that contains this expression (that is, $f \in SemFun$ and $e \in Body(f)$). The basic part of the call graph is built upon the information gathered about static MFA-calls. Namely, if the expression inside f refers to the function f', the call graph contains an edge from f to f'.

$$\frac{e \overset{funref}{\longrightarrow} f'}{f \overset{funcall}{\longrightarrow} f'}\ e \in Body(f)$$

However, in our representation there are other kinds of function calls registered, so to be able to distinguish the different call types, we label the edges of the call graph. Static calls are labelled by *funcall*.

Another group of function calls that we can successfully identify is the unambiguous dynamic call. In such calls the identifiers of the callee may be defined not at the call but at another program part, provided that they are fully calculable with data-flow reaching.

$$\frac{e \overset{dynref}{\longrightarrow} f'}{f \overset{dyncall}{\longrightarrow} f'}\ e \in Body(f)$$

In our call graph the unambiguous dynamic calls are labelled by *dyncall*. These references are as certain as static ones are, that is, neither approximation nor heuristics are applied during analysis.

The third kind of analysed references are ambiguous references. A such function reference always points to an opaque function entity, which is not fully defined. We do not include opaque functions into the call graph, instead, we associate such functions with fully defined ones. The latter relation is called *may_be*. Consequently, the combination of the ambiguous reference and the *may_be* relation determines the ambiguous function calls.

$$\frac{e \overset{ambref}{\longrightarrow} f' \quad f' \overset{may\_be}{\longrightarrow} f''}{f \overset{ambcall}{\longrightarrow} f''}\ e \in Body(f)$$

By applying this rule, a function with an ambiguous call expression will be linked to all the functions the call may refer to. Apparently, a call may only refer to exactly one function, however, static analysis cannot determine which of the ambiguously called functions will be actually called at run-time.

$$\frac{mfa(f) = (\bot, n, a) \vee mfa(f) = (m, \bot, a) \vee mfa(f) = (m, n, \bot) \qquad mfa(f') = (m, n, a)}{f \xrightarrow{may\_be} f'} \; m, n \in Atom, a \in \mathbb{N} \tag{MAY1}$$

$$\frac{mfa(f) = (m, n, i^{\geq}) \qquad mfa(f') = (m, n, j)}{f \xrightarrow{may\_be} f'} \; m, n \in Atom, i, j \in \mathbb{N}, j \geq i \tag{MAY2}$$

4. Use cases in the RefactorErl refactoring tool

By refining the call graph with dynamic invocations, we get a deeper and more accurate insight into the inter-procedural relationships of the system under analysis. While performing different code analysis, refactoring transformations, or cyclic dependency examination, we can utilise the refined function call information. Basically, the refined call relation influences the preciseness of almost every function-related refactoring steps and code analysis (also including code clustering, whose result highly depends on call relations).

Side effect analysis. So far, in the case of expressions containing dynamic calls, we could not decide whether they have side effects or not, since we did not know which function is being called within the expression (potentially having side effects). By default, the Refactorerl tool regards every dynamic call as side effected until any analysis has successfully proved the contrary.

With the new call analysis results we are able to refine the static analysis of side effects. When an unambiguous dynamic call is recognised, we can identify the callee and propagate its side effects related properties. In the example below we have a function named *f* which calls the function *foo* via an apply-call. If *foo* is provably side effect free, then *f* is certainly side effect free as well.

```
f(S) -> apply(m, foo, [S]).
```

Refactoring. The result of the side effect analysis obviously has impact on code refactoring transformations, since they are based on the static code analysis. For instance, when we reorder the arguments of a function, the actual parameters in the calls to this function should also be reordered. However, if any of the parameters might have side effects, we do not allow the transformation, as it might violate the behaviour preservation principle of refactoring (note that in Erlang, the arguments of a call are evaluated strictly from left to right). Now with the help of dynamic function call analysis we can reduce the number of the expressions with indeterminate dirtiness, thus we can allow much more transformations to perform.

As an other example, in case of renaming a function we should replace all occurrences of the old function name with the new one. If we did not recognise the dynamic references of a function, we would not be able to change the function name inside those expressions and consequently we would completely modify the meaning of the program. Listing 9 shows a module whose function *plus* is called dynamically. We rename *plus* to *add* (Listing 10). If we did not have dynamic call analysis, no references would be renamed, resulting in undefined function calls and run-time errors.

5. Conclusions

In this paper we presented that the function analysis and the data-flow analysis can be effectively combined and we also defined how static function call graphs of Erlang programs can be extended with dynamic call references uncovered by use of data-flow analysis. We successfully classified the dynamic call constructs of the language both on the syntax level and based on the way the function identifiers and call arguments are constructed and passed to the call. We defined static and dynamic calls, as well as MFA-calls, apply-calls, whose syntax definition was formally discussed. Also, we formally defined the connection between dynamic call expressions and their callee, and we introduced the concept

Listing 9. Original

```
-module(m).

plus(A,B) -> A + B.

f(A,B) -> apply(m,plus,[A,B]),
          Fun = plus,
          m:Fun(A,B).
```

Listing 10. Renaming plus to add

```
-module(m).

add(A,B) -> A + B.

f(A,B) -> apply(m,add,[A,B]),
          Fun = add,
          m:Fun(A,B).
```

of ambiguous calls. Ambiguous references are not calculable at compile-time and therefore are not fully identified by static analysis, but we presented a method that represents ambiguous calls by opaque function entities and *may_be* relations in order to include them into the call graph. Finally, we precisely formalised how the newly defined references can refine the static call graph.

6. Related results

As far as we are aware of existing static analyser tools for the Erlang programming language, only TypEr (Type Annotator of Erlang Code [8]) extends its call graph — built from static function calls — with a subset of possible dynamic calls. Unlike other Erlang type analyses (e.g. soft-typing by Nystrom [9] and sub-typing by Marlow and Wadler [10]), the success typing method of TypEr is present in the Erlang/OTP environment. In case of dynamic MFA-calls it tries to use the result of a kind of data-flow analysis to gather out the module and function names. In contrast to our analysis method, it extends the call graph with a dynamic call only if both the module and function names are clearly deducible, while apply-calls and ambiguous calls are completely ignored.

Other dynamically typed functional (and scripting) languages provide similar dynamic function call constructs. There are a large number of papers investigating static analysis and typing of dynamically typed languages. For JavaScript [11] as well as for Scheme [12] well-defined typing, data- and control-flow analysis methods have been developed. Nevertheless, to our best knowledge, none of these has detailed the consideration of dynamic call constructs.

In imperative languages, a commonly applied method to construct dynamic calls is using function pointers (variables that point to the address of functions). Building call graphs in the face of pointers requires points-to analysis to provide accurate results, which means we have to estimate the contents of pointer variables by propagating pointer assignments, copies, and arithmetic across program data flows. There is a large body of theoretical work on various pointer analyses [13] and their application for call-graph construction with different degrees of cost and precision [14, 15].

7. Future work

We presented how data-flow analysis can help to refine the function call graph, however, data-flow relations can also be adjusted according to the new information about dynamic calls. Consequently, the data-flow analysis and the function analysis mutually influence each other. It would be interesting to define an iterative algorithm that produces more and more precise data-flow information and call graph, to examine the cost and result of each iteration step, and to give a reasonable termination condition.

When function references cannot be identified with the use of data-flow analysis either, we may apply analyses taking run-time details into account. Possibilities include symbolic evaluation as well as dynamic analysis.

Further development steps could deal with *eval* expressions, which are applicable for evaluating any Erlang code stored in a string. Such

constructs could result in additional dynamic function calls, as the evaluated string may contain function calls to be analysed in some way. The static analysis of *eval* constructs present in Erlang is an open question.

Acknowledgement

We would like to thank Zoltán Horváth, Róbert Kitlei and Melinda Tóth for their help and support given during our research. The project was supported by TECH_08_A2-SZOMIN08.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, July 1999.

[2] "RefactorErl Home Page," 2011, http://plc.inf.elte.hu/erlang.

[3] F. Cesarini and S. Thompson, *ERLANG Programming*, 1st ed. O'Reilly Media, Inc., 2009.

[4] "Open Source Erlang," 2011, http://www.erlang.org.

[5] B. G. Ryder, "Constructing the Call Graph of a Program," *IEEE Trans. Softw. Eng.*, Vol. 5, No. 3, 1979, pp. 216–226.

[6] Z. Horváth *et al.*, "Modeling semantic knowledge in Erlang for refactoring," in *International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009, Selected papers*, ser. Studia Universitatis Babeş-Bolyai, Series Informatica, Vol. 54(2009) Sp. Issue, Cluj-Napoca, Romania, Jul 2009, pp. 7–16.

[7] M. Tóth, I. Bozó, Z. Horváth, and M. Tejfel, "1st order flow analysis for Erlang," in *8th Joint Conference on Mathematics and Computer Science, MACS 2010*, 2010.

[8] T. Lindahl and K. Sagonas, "Typer: a type annotator of erlang code," in *Proceedings of the 2005 ACM SIGPLAN workshop on Erlang*, ser. ERLANG '05. New York, NY, USA: ACM, 2005, pp. 17–25. [Online]. http://doi.acm.org/10.1145/1088361.1088366

[9] S.-O. Nyström, "A soft-typing system for Erlang," in *Proceedings of the 2003 ACM SIGPLAN workshop on Erlang*, ser. ERLANG '03. New York, NY, USA: ACM, 2003, pp. 56–71. [Online]. http://doi.acm.org/10.1145/940880.940888

[10] S. Marlow and P. Wadler, "A practical subtyping system for Erlang," *SIGPLAN Not.*, Vol. 32, August 1997, pp. 136–149. [Online]. http://doi.acm.org/10.1145/258949.258962

[11] S. H. Jensen, A. Møller, and P. Thiemann, "Type Analysis for JavaScript," in *Proc. 16th International Static Analysis Symposium, SAS '09*, ser. LNCS, Vol. 5673. Springer-Verlag, August 2009.

[12] O. Shivers, "Control-Flow Analysis in Scheme," in *PLDI*, 1988, pp. 164–174.

[13] B. Steensgaard, "Points-to analysis in almost linear time," in *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ser. POPL '96. New York, NY, USA: ACM, 1996, pp. 32–41. [Online]. http://doi.acm.org/10.1145/237721.237727

[14] E. Horváth, I. Forgács, Ákos Kiss, J. Jász, and T. Gyimóthy, "General flow-sensitive pointer analysis and call graph," in *Proceedings of the Estonian Academy of Sciences, Engineering*, Vol. 11, December 2005, pp. 286–295.

[15] A. Milanova, A. Rountev, and B. G. Ryder, "Precise call graph construction in the presence of function pointers," In Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation, Tech. Rep., 2001.

e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 77–85, DOI 10.5277/e-Inf130108

Software Engineering Team Project – lessons learned

Bogumiła Hnatkowska*

*Faculty of Computer Science and Management, Institute of Informatics, Wrocław University of Technology

bogumila.hnatkowska@pwr.wroc.pl

Abstract

In the 2010/11 academic year the Institute of Informatics at Wroclaw University of Technology issued 'Software Engineering Team Project' as a course being a part of the final exam to earn bachelor's degree. The main assumption about the course was that it should simulate the real environment (a virtual IT company) for its participants. The course was aimed to introduce issues regarding programming in the medium scale, project planning and management. It was a real challenge as the course was offered for more than 140 students. The number of staff members involved in its preparation and performance was more than 15. The paper presents the lessons learned from the first course edition as well as more detailed qualitative and quantitative course assessment.

1. Introduction

Most of the Technical Universities offer a course 'Team project' for the under graduate students but rather rarely such a course is a part of final exam to earn bachelor's degree.

Polish Ministry of Education published 'Standards of education for Computer Science faculty (Bachelor studies)' in 2007 [1]. The document not only forced the Universities to offer the course mentioned above, but also gave the opportunity to treat it as a part of bachelor's thesis. The Institute of Informatics at Wroclaw University of Technology seized this chance and included 'Software Engineering Team Project' in the set of obligatory courses. The course was issued first time in the 2010/11 academic year in the winter semester. However, the course was elaborated based on experiences gained from others, previously offered, courses.

The aim of the paper is to present the lessons learned from the first course edition as well as more detailed qualitative and quantitative course assessment. The gained experiences could be helpful in organizing similar courses, and the description could serve for comparison purposes.

The paper is organized as follows. Section 2 presents the course assumptions, while Section 3 – the process of course preparation. Section 4 gives a detailed description of the course implementation. The course evaluation is the main part of Section 5. It presents valuable statistics, interesting observations and lessons learned. The last Section 6 summarizes the paper.

2. Course design

The main assumption about the course was that it should simulate the real environment (a virtual IT company) for its participants. The course was aimed to introduce issues regarding programming in the medium scale, project planning and management. It was a real challenge as the course was offered for more than 140 students. The number of staff members involved in its preparation and performance was more than 15.

The aim of the course was to develop something useful for a customer (usually a software product), and possibly to deploy it in a target environment. The course was thought as a final checking of the effects of previously offered courses, as it demanded the broad knowledge of software engineering.

According to [2] the Software Engineering Team Project was placed at the 3rd level of maturity:

Defined. Class projects are undertaken based on a defined, documented, and standardized process. The instructor relies on an established process which serves as a foundation for further process improvement. Course practices can be consistently implemented, regardless of the presence or absence of certain key instructors.

We have also implemented some practices from the 4rd maturity level [2]:

Quantitative Managed. Course projects are quantitatively measured using statistical methods. Measurements on student projects are gathered and analyzed for further improvement.

The course schedule was planned for only 10 weeks of very intensive work. A student was given 15 ECTS points for the course, what was translated into 40 hours per week of work for each course participant (9 hours at the Institute, and 31 hours for the private study). No other forms than project were involved in the course. Course design was prepared and is further presented according to the suggestions given in [3]. The description considers following issues:

- team formation,
- team supervision,
- problem statement and assignment,
- team communications,
- team assessment,
- development process.

2.1. Team formation

There are two possibilities for team formation [3]:

1. students create teams by themselves,
2. students are assigned to the teams.

We followed the first scenario, as it was easier from managerial point of view. The students were expected to organize themselves into four people teams (exceptionally, 3 or 5 people were allowed). The team was organized based on student's previous experiences. The general assumption was that the team is self-organizing with inter-changeable roles.

2.2. Team supervision

To simulate the reality, students should have a considerable amount of freedom. On the other hand, since students usually had no project experience, some amount of supervision, monitoring and guidance was needed to ensure sufficient progress and a successful result [3]. It was the responsibility of a team supervisor. Each supervisor looked after 1–4 teams (in one case 8 teams).

Project's supervisor played many roles. First of all his/hers responsibility was to formulate a project subject and its requirements (both functional and non-functional). In that way the person became a product owner. He or she was also responsible for steering the project complexity. The supervisor was allowed to reduce the scope (if it occurred unrealistic) or to extend it (if the capability of the team was greater). He/she also decided about final product assessment. Additional responsibility of the person was to give continuous feedback to the teams.

2.3. Problem statement and assignment

Each team selected a subject of the project from the list of themes, proposed by academic teachers playing the role of supervisors. Teams were also allowed to propose something interesting by themselves (the proposals were assessed by the supervisors, and after acceptance were realized). As the course was a part of the final exam, all themes had to be accepted by Department Authorities before the start of classes.

2.4. Team communications

Team members needed to communicate quite often. The responsibility of the University was to prepare the conditions for that. Depending on the type of project and the decision of the supervisor the team met at the University (orga-

nized lessons): once per week for 9 hours, twice a week for 4,5 hours or three times a week for 3 hours. In one classroom at most 4 teams were working in the same time. Each had access to computers, and a blackboard. In [3] there was a suggestion, that each team should create and regularly maintain a web page, and have documents repository.

2.5. Assessment

The supervisor was responsible for determining student's final grade. There are two common approaches to students' assessment [3]:

1. to give each team member the same grade and,
2. to give each team member a grade based on his individual contribution.

We combined both of them. In general the whole team obtained the same grade. It was offered basing on product quality, documentation quality, and the product presentation. The main focus was put on the final product itself. Its presence was a necessary condition to pass the course. The second element taken into account was the student engagement. To motivate the students to better work, the supervisor was allowed to give a student so called yellow card to indicate that his/her engagement in the project was too low, and the quality of his/hers artifacts were bad. The first yellow card decreased the final grade by 1, while the second yellow card became a red card and meant that the student failed the course.

We pondered over including peer to peer assessment component to the student's assessment recommended by many authors [3, 4], but finally we rejected this idea. We had bad experiences in using it in past projects. The students did not want to fairly judge each other. Similar results of formally conducted experiments were reported in [5].

2.6. Development process

The course assumption was that a development process should involve all necessary stages of software development from requirements to implementation and testing. A stage of project deployment was welcomed. The selected development process should fit to different projects' characteristics.

3. Course preparation

Preparation for the course was rather a long process which took about 6 months. The authorities of Institute of Informatics appointed 8 people team for that purpose. The staff members team involved experts from different domains: software engineering, data bases, computer networks, programming languages, and net administrators. The team consisted of people with experiences in providing team projects before, what resulted in some publications, e.g. [6].

The main task of the team was to develop course's recommendations including methodologies (both development and managerial), best practices, and supporting tools. It was expected that the team will also prepare educational materials for both: the faculty staff members, and students attending the course.

The task was challenging of many reasons: (1) very short time of project realization (only 10 weeks), (2) diversity of projects' subject (the recommended tools, techniques and practices must have been adjusted to all of them).

The outcomes of the course preparation process included a report, published as an internal document by the Institute of Informatics, covering all the topics mentioned above as well as some publications presented on the National Conference in Software Engineering [7], and on the International Conference ISAT [8].

The main recommendations were as follows:

– methodologies: Scrum methodology was selected as a basic one, as it fited to different kinds of projects which could be done in an iterative manner. The team considered several methodologies, both agile (Scrum, XP) and heavier ones e.g. UP, RUP. The UP methodology was recommended only for typical software development projects, for which the strong need for documentation existed. The

assumption was that each project should use at least: Product backlog, and Sprint backlog [9]. The Sprint backlog should contain tasks together with time evaluation (for at least 90% of sprint capability, about 30 h a week for each team member). The other artifacts (e.g. specifications, documentations etc.) were under supervisor jurisdiction,
- best practices: The staff team analyzed the best practices for all above mentioned software development processes, trying to define a minimal set of the most useful ones (obligatory practices). The selected practices included: iterative development (sprints), and self-organizing team with daily meetings [9]. Since each development process includes requirement specification, the requirements were presented with the use of product backlog. Requirement analysis, if it is possible, should be based on use-cases specification. The coding standards should be used at implementation stage. The other recommended (but not obligatory) practices included: source code sharing, TDD development, and continuous integration,
- tools: Team Foundation Server (TFS) [10] was selected as a primary supporting tool, mainly because the University obtained professional support from the Microsoft Polska Company; the tool allows to plan releases, sprints, and tasks according to Scrum methodology; it offers also other useful capabilities, as source code management system, or continuous integration support. It integrates well with Microsoft Visual Studio and good enough with Eclipse platform. Moreover, it offers a web site for each projects, and is a good mean for communication purposes (e.g. Wiki). The functionality is available by the Internet. Students were also allowed to use virtual machines to deploy their solutions.

Before running the course, the staff team presented the assumptions about it to all members of the Institute, e.g. organization details. Moreover, a training covering the usage of the Team Foundation Server in the context of Scrum methodology was conducted.

4. Course implementation

The course was attended by 37 groups (146 students) supervised by 17 people. Each supervisor, as a product owner, set a number of releases, a number of sprints, and a length of sprints. Mostly, the projects had one release (66%), or two (32%). One project had three releases (2%). Sprints lasted 7 days (89%) or 14 days (11%). The capacity for 7 days iterations was calculated usually for 30h/iteration/person (10 teams) or 40h/iteration/person (27 teams) – it was under supervisor jurisdiction.

The subjects were very diverse, e.g. Conference management system, Team competition management system, System for collection and analyzing samples of handwriting, Virtual campus of Wroclaw University of Technology, Collision-free motion of a group of autonomous vehicles – an algorithm using the Webots environment. A few subjects came from industry, e.g. System for support decision making based on data collected in a data warehouse. The results of some projects are visible in the Internet: [11–13].

5. Course assessment

5.1. Quantitative assessment

To compare different teams, and to find out existing shortcomings a quantitative assessment of the course was done. The measure values were taken from the TFS (product backlog, and sprint backlog) which was the obligatory managerial tool for all teams, and their supervisors. However, one team of unknown reasons did not use the TFS, so it was excluded from further consideration. The research questions were formulated as follows:
- What was the number of product backlog items (PBI), and sprint backlog items (tasks)?
- How many hours did the teams spend on tasks realizing during the whole project?
- What was the coverage (in percent) of the declared capacity?

Statistic	The number of PBI	The number of tasks	The number of work hours	Capacity coverage [%]
Average	22,68	128,33	792,50	0,77
Minimum	4,00	32,00	40,00	0,04
Maximum	89,00	313,00	1698,00	1,77
Standard deviation	19,53	68,76	375,76	0,36

Table 1. The basic measures for projects obtained from TFS tool

I wanted to establish the parameters of a 'typical' project which could serve as a reference for other projects. The answers for the questions are presented in Table 1.

The number of PBI fluctuated from 4 to 89, with the average 22,68. Lower numbers of PBI usually meant that the granularity of them was bigger. In many cases where the PBI number was below 10, a development process was rather a waterfall than iterative one with PBI elements representing requirement specification, analysis, design etc., and the tasks associated with them were done in many sprints.

The number of tasks for a project ranged from 32 to 313 (the average was equal to 128,33). Lower number of tasks usually meant that they were very long (sometimes they were estimated for more than 30h) or that most of the tasks were not documented in the TFS (for a team with 32 tasks, the coverage of the declared capacity was 23%, and for another team with 37 tasks, the coverage was only 0,04%). A team with the highest number of tasks (313) defined – during the whole project – a lot of small (2–4 hours) tasks. The coverage of the declared capacity fluctuated from 0,04% (for a team that defined only 37 tasks for 40 hours) to 170% (for a team with the tasks estimated for 1698 hours). Probably, in the former case, the team did not use the TFS tool, while in the later case, the duration of tasks were overestimated; half of them lasted more than 8 hours, many – 20 hours. The average of the capacity coverage was equal to 0,77%. The number of teams with the capacity in the range [80%;120%] was equal to 17, only 2 teams defined the tasks for more than 120%.

The additional research questions were about other good practices the teams could use, mainly, how many teams used the source control version tool, integrated with the TFS? (20 teams, 54%), how many teams registered bugs in the TFS? (8 teams, 21%, while only 5 had more than 3 bugs), how many teams used automatic tests? (5 teams) It should be mentioned that the data were collected only from the TFS tool. From the qualitative assessment of the course I know that some teams used another system of version control, e.g. Git.

Because in [3] there is a suggestion to make available some projects as good examples, I tried to classify projects (product bakclog, sprints, strpint backlog) stored into the TFS into three groups of quality: (1) target (2) minimal (3) non-accepted.

To do that a checklists with disqualifying questions were defined. Below the checklist to disqualify a project to have a target quality is presented:

1. PBI not associated with iterations,
2. Tasks not associated with PBI items, or some sprints without the tasks at all,
3. Tasks without time evaluated,
4. Very long tasks – lasted more than 30h,
5. Mistakes in PBI and/or tasks descriptions, e.g. many repetitions of the same PBI definition,
6. PBI realized in waterfall manner (tasks associated with PBI were realized during many sprints),
7. Capacity coverage outside the range [70%; 130%],
8. States of sprints and tasks not changed properly.

The elements disqualifying a project to have a minimal quality are as follows:

1. 1–5 as for target quality,
2. Capacity coverage outside the range [50%; 150%].

According to the rules defined above we had 6 projects with target quality, 17 – with mini-

Statistic	Course usefulness	Quality of supporting materials	Course organization
Average	4,66	3,37	4,32
Minimum	2,00	1,00	1,00
Maximum	5,00	5,00	5,00
Standard deviation	0,65	0,96	0,82

Table 2. The assessment of course components steaming from closed questions

mal quality, and 14 – with non-accepted quality. The number of bad projects seems to be too big, but I hope that in the next edition this number will decrease as there are some good examples.

The conclusion from the facts presented above is that the supervisors need to better monitor the usage of the TFS tool. The limits (min, and max) for the number of hours for a person/week must be defined and obeyed – the recommended value is 30. The maximal duration for task estimation (8 hours) must be strongly obeyed.

5.2. Qualitative assessment

To perform qualitative assessment of team project a questionnaire for the students were prepared. The questionnaire questions were as follows:

1. (Closed question): Assess the following elements in the scale from 1 (the worst mark) to 5 (the best mark):
   - Course usefulness: ...
   - Quality of supporting materials: ...
   - Course organization: ...
2. (Open question) Which elements of the course were the most valuable?
3. (Open question) Which elements of the course need to be improved/changed in the next issues?

The questionnaires were filled after the final exams, so telling the truth, we asked bachelors about their opinion. The questionnaires were available in two forms: paper and digital one (published in the Internet). We gained the answers from 93 students from 146 who managed to finish the course (what gives 64% response rate). The answers for the closed questions are presented in Table 2.

The grades for all components were rather high – the highest for course usefulness, the smallest for course supporting materials. It is something that should be improved in the next edition.

The questionnaire consisted mainly of open questions. It was a challenge to analyze them. I had defined some synonyms that were present in different answers, and next counted the numbers of people, who used these synonyms in their answers. The results are presented below. The threshold, below which the answers are not presented, was set to 10.

Figure 1 presents how many people pointed out a specific element to be the most valuable.

As it is easily to observe, the team work (39 answers) and agile methodology (38) were perceived as the best choices. An opportunity to acquaintance the new technologies (17) and to put different pieces together (to build the whole project from the beginning to the end – deployment, 17 answers) was inspiring experience. The students appreciated also regular meetings as the means that motivated them to hard work (14) – they needed to answer 3 questions at the beginning of each meeting (what was done, what problems were encountered and what is going to be done to the next meeting). The team members also thanked for interesting themes that allowed them to deeply involve to the project (11).

Of course not all elements were the full success. Figure 2 presents the elements that need to be improved.

First of all the students complained about the short time of the project (24 answers). The Project Management Course was run in parallel with the team project. The students reasonably suggested to move Project Management Course one semester back. They also found TFS environment quite difficult and suggested to do a

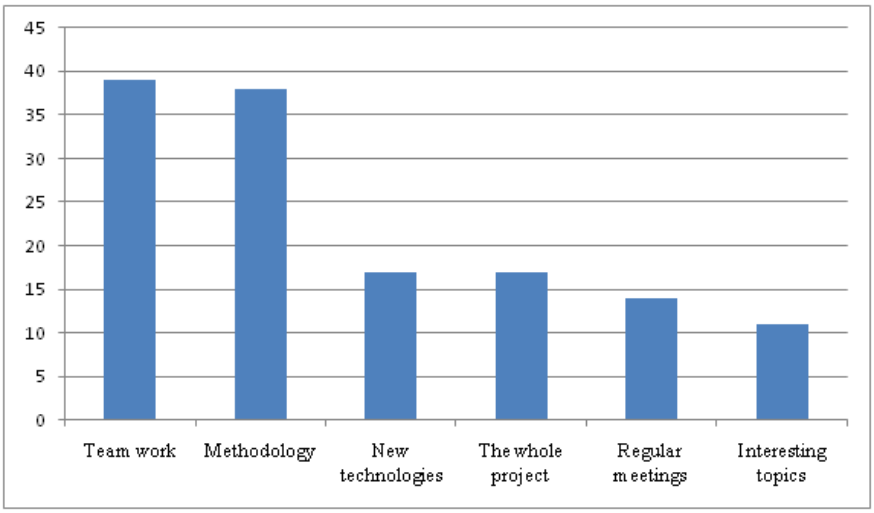

Figure 1. Number of answers for the questionnaire question nr 2

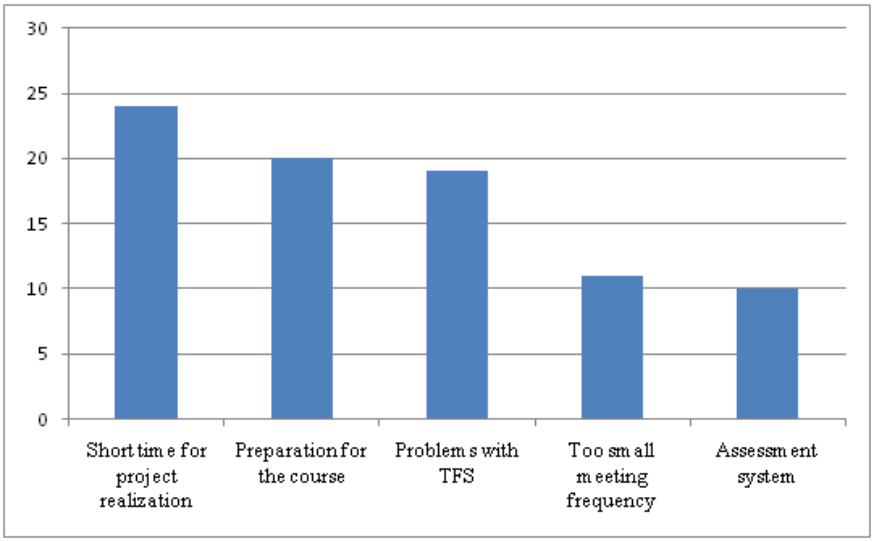

Figure 2. Number of answers for the questionnaire question nr 3

training before vacations (20). The access to the TFS was not such efficient the students expected (19). The students wanted also to meet more often (this opinion was given by the team members who had organized meeting only 1 per week). The students suffered a little from not clear enough assessment system (10).

The same questions were asked to the academic teachers playing the role of projects' supervisors. But the results were very similar to those obtained from the students.

Below a list of possible improvements is presented.

Course preparation:

- Project Management course should be moved to the 6th semester,
- Training of the tools used within a course should be provided in the 6th semester of during vacations,
- Supporting materials should be improved.

Supporting tools:

- Performance parameters (throughput, response time) of the TFS should be better,
- Opportunity of usage of another popular supporting tool should be considered.

Course organization:

– Give the students a longer time for their own topics formulation,
– Attract more topics from the industry,
– Make an assessment system more transparent.

6. Conclusions

The Software Engineering Team Project starting from 2010/11 academic year is a compulsory subject to all students of Informatics at Wroclaw University of Technology. Many universities offer a team engineering course as a part of the studies, e.g. [3, 14–17], but there are some important differences between them and our proposal:

– Course outcome: system prototype versus a working release of a system, often with its installation version and a user manual,
– Focus: educational aims versus put things together (part of the final exam),
– Methodology: due to documentation reasons rather heavier methodologies versus as light methodology as possible,
– Accompanying courses: lecture or seminars versus none.

We found the first course edition as a success, however some elements could be improved. The weakest part of the course was the assessment. More strict rules for it should be defined, especially taking into account the number of working hours. We are going to introduce an obligatory presentation of the project before the members of faculty staff (the last time it was under a supervisor jurisdiction). We are also going to reward the best project with the highest grade (5.5). At that moment we are preparing to the second edition of the course for more than 160 students, e.g. a new version of training materials is under development. The number of topics coming from industry is a little bit higher than the last year. We hope that we manage to eliminate shortcomings and not to lose good points.

References

[1] Rozporządzenie ministra nauki i szkolnictwa wyższego z dnia 12 lipca 2007. (2007). [Online]. http://www.bip.nauka.gov.pl/_gAllery/21/97/2197/20070712_rozporzadzenie_standardy_ksztalcenia.pdf

[2] J. Collofello and C. H. Ng, "Assessing the process maturity utilized in software engineering team project," *Journal of Engineering Education*, Vol. 90, No. 1, 2001, pp. 75–78.

[3] M. Bielikova and P. Navrat, "Experiences with designing a team project module for teaching teamwork to students," *Journal of Computing and Information Technology*, Vol. 13, No. 1, 2005.

[4] N. Clark, P. Davies, and R. Skeers, "Self and peer assessment in software engineering projects," in *Proc. of 7th Australasian Computing Education Conference (ACE 2005)*, D. T. A. Young, Ed. CRPIT, 2005, pp. 91–100.

[5] N. Herbert, "Quantitative peer assessment: Can students be objective?" in *Proc. of 9th Australasian Computing Education Conference (ACE 2009)*, S. Mann and S. Mann, Eds. CRPIT, 2009, pp. 63–71.

[6] I. Dubielewicz and B. Hnatkowska, "Improving software development process implemented in team project course," in *Proc. of the 8th International Conference on Computational Science, Part II.* Berlin: Springer-Verlag, 2008, pp. 687 – 696.

[7] ——, "Praktyki w inzynierii oprogramowania – perspektywa pracy zespolowej," in *Inzynieria oprogramowania w procesach integracji systemow informatycznych*, C. O. Janusz Gorski, Ed. Gdansk: Pomorskie Wydawnictwo Naukowo-Techniczne PWNT, 2010, pp. 121–228.

[8] ——, "Best practices in students team projects," in *Information systems architecture and technology: IT models in management process*, Z. Wilimowska, Ed. Wroclaw: Oficyna Wydawnicza Politechniki Wrocławskiej, 2010, pp. 487–497.

[9] R. Levin. Understanding scrum – best practices guide. (2010). [Online]. http://www.brighthub.com/office/project-management/articles/68791.aspx

[10] J. Meier, J. Taylor, P. Bansode, A. Mackman, and K. Jones. Team development with visual studio team foundation server. (2007, Sep). [Online]. http://msdn.microsoft.com/en-us/library/bb668991.aspx

[11] Collision-free motion of a group of autonomous vehicles – an algorithm using the webots environment. (2011). [Online]. http://www.youtube.com/watch?v=tzm2uU8_nQA

[12] Laboratory of the distributed computer networks portal. (2011). [Online]. http:

//156.17.130.12/Main.aspx

[13] Virtual campus of wroclaw university of technology. (2011). [Online]. http://www.ii.pwr.wroc.pl/WirtualnyKampusPWr/index.html

[14] D. Delaney and G. Mitchell. Tutoring project-based learning: a case study of a third year software engineering module at nui. (2005). [Online]. http://www.aishe.org/readings/2005-2/contents.html

[15] G. Dobbie and G. Bartfai, "Teaching software engineering in a computer science department," in *Proc. of International Conference Software Engineering: Education and Pracitce*, Dunedin, New Zealand, 1996, pp. 58–63.

[16] Unit of study engg1805 professional engineering & it. (2011). [Online]. http://sydney.edu.au/engineering/it/~engg1805/Documents/ENGG1805CourseOutline.pdf

[17] M. Zaigham, "A framework for software engineering education: A group projects approach," *International Journal of Education and Information Technologies*, Vol. 1, 2007.

e-Informatica Software Engineering Journal (EISEJ) is an international, open access, peer-reviewed journal that concerns theoretical and practical issues pertaining development of software systems. Our aim is to focus on experimentation and data mining in software engineering.

The purpose of **e-Informatica Software Engineering Journal** is to publish original and significant results in all areas of software engineering research.

The scope of **e-Informatica Software Engineering Journal** includes methodologies, practices, architectures, technologies and tools used in processes along the software development lifecycle, but particular stress is laid on empirical evaluation.

e-Informatica Software Engineering Journal is published online and in hard copy form. The online version (which is our primary version) is open access, which means it is available at no charge to the public.

Topics of interest include, but are not restricted to:

— Software requirements engineering and modeling
— Software architectures and design
— Software components and reuse
— Software testing, analysis and verification
— Agile software development methodologies and practices
— Model driven development
— Software quality
— Software measurement and metrics
— Reverse engineering and software maintenance
— Empirical and experimental studies in software engineering (incl. replications)
— Evidence based software engineering
— Systematic reviews and mapping studies
— Meta-analyses
— Object-oriented software development
— Aspect-oriented software development
— Software tools, containers, frameworks and development environments
— Formal methods in software engineering.
— Internet software systems development
— Dependability of software systems
— Human-computer interaction
— AI and knowledge based software engineering
— Data mining in software engineering
— Prediction models in software engineering
— Tools for software researchers or practitioners
— Project management
— Software products and process improvement and measurement programs
— Process maturity models
— Search-based software engineering

Papers can be rejected administratively without undergoing review for a variety reasons, such as being out of scope, being badly presented to such an extent as to prevent review, missing some fundamental components of research such as the articulation of a research problem, a clear statement of the contribution and research methods via structured abstract or the evaluation of the proposed solution (empirical evaluation is strongly suggested).

The submissions will be accepted for publication on the base of positive reviews done by international Editorial Board and external reviewers.

English is the only accepted publication language. To submit an article please enter our online paper submission site.

Subsequent issues of the journal will appear continuously according to the reviewed and accepted submissions.

http://www.e-informatyka.pl/wiki/e-Informatica

e-Informatica

ISSN 1897-7979