SilabMDD - A Use Case Model Driven Approach

Dušan Savić, Siniša Vlajić, Saša Lazarević, Vojislav Stanojević, Ilija Antović, Miloš Milić[1], Alberto Rodrigues da Silva[2]

Faculty of Organizational Sciences, University of Belgrade [1]
Department of Computer Science and Engineering
IST / University of Lisbon[2]

Abstract - *Model-Driven Development (MDD) is a software development paradigm that emphasizes the importance of using models during the entire software development process, models with different levels of abstraction. In our SilabMDD approach, use cases models allow to define user and software requirements. These models are specified in the SilabReq language which is implemented inside JetBrains Meta Programming System (MPS) and can be used as plug-in for IntelliJ IDEA or for the MPS tools.*

1. INTRODUCTION

The development of information systems is a complex and social process because it involves many interactions among different stakeholders. In order to make this process successful it is necessary to understand the system requirements and document them in a suitable manner. There are multiple definitions for requirements, namely: (1) a property that must be exhibited in order to solve some real-world problem [1]; (2) needs and constraints placed on a software product that contribute to the solution of some real-world problem [2]; (3) a condition or capability needed by a user to solve a problem or achieve an objective; or (4) a condition or capability that must be met or processed by a system (or system component) to satisfy a contract, standard, specification, or other formally imposed documents [3]. Additionally, there are many forms for requirements presentation such as natural language, constrained natural language or model based requirements language [4].

Requirements engineering (RE) involves two main processes [4]: (1) requirements development (with elicit, analyze, specify, and validate software requirements) and (2) requirements management process. Many RE approaches have been discussed in the literature, which differ in their methods, modeling techniques and modeling languages. Widely accepted approaches in 70ies and 80ies were mainly data and functional-oriented analysis techniques, while object-oriented approaches were emerged and were popular during the late 80s and 90s. Another approach emerged more recently were goal-oriented requirements engineering [5]. Common to all these approaches, particularly in their early period, is the clear separation of RE process from software development process.

Other software paradigm, referred to as Model-Driven Development (MDD) [6], is a software paradigm that emphasizes the importance of models. The aim of MDD is to use models throughout the software development process at different levels of abstraction. Therefore, models are not used only to document some part of a system; models are first-citizen in software development. MDD processes usually start to develop a requirements model which is defined to describe user's needs in a computational independent way. Then, this model can be refined into one or more models that describe the system without considering technological aspects. Finally, these models are either refined into design models (that describe the system by using concepts of a specific technology) and are then translated into a source code; or are directly derived to a code if they contain enough information to implement the software system in a precise and complete way [7].

However, despite the importance of RE as a key success factor for software development projects there is still a lack of MDD methods that would cover the full development lifecycle, from a RE level to a development level with source code generation or writing activities [8] [9].

In this paper we introduce SilabMDD approach that is a use case and MDD approach that use SilabReq as a use cases specification language. Furthermore, we present how SilabReq is supported by JetBrains Meta Programing System (MPS). The goal of SilabMDD is to provide a complete software development workbench to be used by requirements engineers, developers, as well as by non-technical stakeholders.

This paper is organized as follows. Section 2 describes the background of this work. Section 3 presents SilabMDD approach while Section 4 concludes the paper and outlines future work.

2. BACKGROUND

Requirements are mostly documented using natural language. However, natural language requirements specification tends to be ambiguous, unclear, and inconsistent [4]. On the other hand, documenting requirements using semi-formal models require using specific modeling language for each particular perspective. Pohl proposed three types of requirements proposed by [4]: (1) goals, which document intentions of stakeholders; (2) scenarios, that describe concrete example of system usage; and (3) solution-oriented requirements, that can be used as complement each other. Different requirements modeling languages can be used for modeling different requirements artifacts for different perspective. For example, i* [4] and KAOS[4] goal oriented models can be used for specification of the goals; UML use case, sequence and activity diagram can be used for modeling scenarios; while UML state machines or data flow daigrams can be used for modeling system's behavior.

The specification of requirements is a difficult task because requirements are read by many participants of the software development process with different technical knowledge. People prefer to use textual specification of requirements, but their representations are not suitable for automatic validation, transformation and even reusing. We need a structured language for requirements specification that will be understandable by most of these participants but also will be precise enough to enable automatic validation and transformation. This language should be defined by meta-modeling or grammar ware in order to enable automatic or semi-automatic processing.

UML is a standard language for modeling software systems and many people have also used it for requirements specification. However, some authors have argued that UML has some deficiencies as a semiformal requirements specification language [10].

There are other Requirements Specification Language (RSL) that use natural language in a controlled way. Smiałek at el. defined RSL as a semiformal natural language that employs use case for specifying requirements [11]. Each scenario in a use case contains special controlled natural language SVO (O) sentence. RSL has been developed as a part of ReDSeeDS project [12]. ReDSeeDS approach covers a complete chain of model-driven development – from requirements to code[13] .

The goal of ProjectIT [14] [15] is to provide a complete software development workbench, with support for project management, requirements engineering, analysis, design and code generation activities. ProjectIT-Requirements is the component of the ProjectIT architecture that deals with requirements issues. The main goal of the ProjectIT-Requirements is to develop a model for the definition and documentation of requirements, which, by raising their specification rigor, facilitates the reuse and faster the integration with MDD development environments driven by models. Taking into account the different types of requirements, this project uses software requirements, those that can more easily be "converted" in software design models by MDD approaches [16] .

Recently, the ProjectIT's RSL evolved to a more flexible approach named RSLingo[17][18]. RSLingo is a linguistic approach for improving the quality of requirements specification, which is based on two languages and mapping between them: the RSL-PL and the RSL-IL. RSL-PL (Pattern Language) is an extensible language for defining linguistic patterns dealing with information extraction from requirements written in natural language. On the other hand, RSL-IL (Intermediate Language) is a formal language with a fixed set of constructs for representing and conveying RE-specific concerns.

3. THE SILABMDD APPROACH

SilabMDD approach emerged as a key result of Silab Project which was initiated in 2007 in the Software Engineering Laboratory at Faculty of Organizational Sciences, University of Belgrade. The main goal of this project was to enable automated analysis and processing of software requirements in order to achieve automatic generation of different parts of a software system. In the beginning, Silab Project has been divided in two main sub-projects SilabReq and SilabUI that were being developed separately. Initially the SilabReq project focused on the formalization of user requirements and their transformations to different UML models to facilitate the analyses process and to assure the quality of software requirements. On the other hand, SilabUI project focused on automatic generation of user interfaces from use cases specifications. When both subprojects reached the desired level of maturity, they were integrated in a way that the results of SilabReq were used as input for SilabUI project. As a proof of concept, Silab project was used for the Kostmod 4.0 [19] project, which was implemented for the needs of the Royal Norwegian Ministry of Defense.

After several years of using this project in developing different intensive software system we are established a SilabMDD approach. This section introduces the conceptual view of the SilabMDD approach.

A. Overview of the SilabMDD approach

Fig. 1 depicts the key artifacts of SilabMDD approach. SilabMDD approach is a use case and model driven approach.

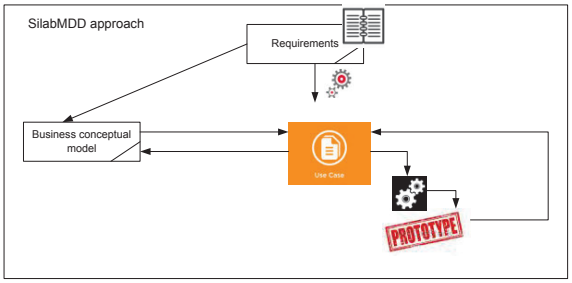

Figure 1. Modeling circle in SilabMDD approach

Usually in MDD the source code is (semi) automatically generated from the models. Despite the fact that use cases are narratives, there is no single standard that specify what textual specification of use case should be. SilabMDD approach includes SilabReq language that is a DSL used for use case specification. SilabReq language allows requires a rigorous definition of the use case specification, particularly description of sequences of action steps, pre- and post-conditions, and relationships between use case and elements (classes) defined in domain models.

SilabMDD approach is also language-oriented. Language Oriented Programming [20] presents a style of development which operates about the idea of building software around a set of DSL, while Language

Workbench is a generic term for tools that use this style of programming. Meta Programming System[1] (MPS) from JetBrains[2] is one of the most popular meta-programming tool that enables language oriented programming with a projection editor in persistent abstract representation that was used to develop SilabMDD's languages.

SilabMDD is use-case driven approach but it do not pay much attention to the way in which use-cases are elicited. They can be derived from business process or from text requirements. If requirements are expressed in some form of model as in RSLingo (using RSL-IL[17]) it is possible to automatically use appropriate transformation to deliver use cases. Integrated in this the specification process, use cases can be specified using SilabReqUC language and continuous inspection of business model. For the description business conceptual model we propose the SilabReqBCM language. Use case actions, pre-conditions and post-conditions are specified in the context of business conceptual models. Therefore, SilabMDD approach use SilabReqUI language which is primary use for specification user interface prototype.

B. *Specification of use case from different level of abstraction*

The SilabReq allows defining use cases specifications at different levels of abstractions according to the different roles involved on this process . There are three different abstraction levels: (1) *use case interaction level* (high-level), (2) *use case behavior level* (medium-level), and (3) *use case UI-based level* (lower-level) [21][22]. Each abstraction level extends and semantically enriches the previous level. Actually, we can use the same model for both user and system requirements. Transformations among these different levels are used to create multiple views as well as for code generation.

Use case action can divide in two categories: (1) the actions performed by the users and (2) the actions performed by system [21,22]. Both categories contain different types of actions. In the category in which actions are performed by the user, we have identified actions types such as: (1.1) Actor Prepare Data to execute System Operation (APDExecuteSO) and (1.2) Actor Calls System to execute System Operation (ACSExecuteSO). On the other hand, in the category in which actions are performed by the system, we have identified two action types such as: (2.1) System executes System Operation (SExecuteSO) and (2.2) System replies and returns the Result of the System Operation execution (SRExecutionSO).

The main task that use cases have *at the highest level of abstraction (interaction level) is user requirements specification*. Therefore, this level allows non-technical stakeholders to quickly read and understand use case descriptions. Use cases alone are not sufficient for user requirements specification. Therefore, use cases are complemented with glossary and business rules. Glossary and business rules are specified within the same language (but it is possible to create and use other specific languages). They are specified separately, but connected with some elements of use cases such as pre-conditions, post-conditions or use case actions. Business rule and terms (in the glossary) are specified with unique identification, name and description. At this level of abstraction, each use case specification consists in the following elements (see Fig.2): unique use case identifier, use case name, the actors who participate in use case, the business entity over which the use case is executed, main and alternative use case scenarios, and use case pre-conditions and post-conditions.

Business rules are used for specification of use case pre-conditions and post-conditions. *Pre-conditions* and *post-conditions* are specified in the context of the system state as a pair of entities and its state. In pre-condition, the system state defines the conditions that must be satisfied before use case starts. This business rules are specified in context of business domain model. Therefore, before use case starts, business entity over which the use case is executed and related entity must be in some specific state. For example, the user must be logged in as administrator (user as an entity and login as a state), the order must exist (order as entity, exist as state), the form for creating bill is open and the order exist (form for bill as state and open as state, order as entity and exist as state). The similar situation is applied to post-conditions specification. After successful execution of a use case, its entity of the system will be in some particular state (for example order as entity will be saved). Action business rules are used to specify data entered by actor (choose or select), or data returned from system. At this level of abstraction, these rules are related with APDExecuteSO action and SRExecutionSO action. Both of these actions are specified in context of business domain model. The Fig. 2 describes the use case specification template document, entity and business rule specification document. The specification document looks like a wiki based document in the way that it is possible to navigate through document.

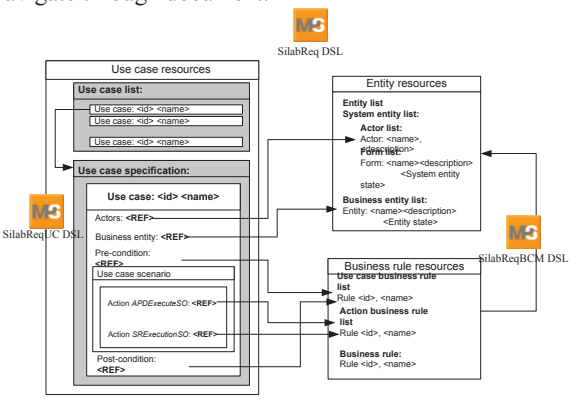

Figure 2 Use case specification from user perspective

The specification of use cases at the medium level is used to determine the desired functionality of the system. At this level, a use case scenario specification is extended with the specification of ACSExecuteSO and SExecuteSO

[1] http://www.jetbrains.com/mps/
[2] http://www.jetbrains.com/

actions. These use case actions are used to define a function that a system should provide. Fig.3 describes how medium-level use case specification extends the high-level specification. This figure describes part of previous template document, which is extended with the specification system operations that contains: system operation pre-condition, successful and error system response, as well as system operation post-condition.

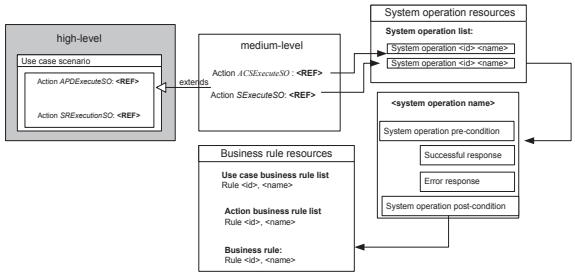

Figure 3 Use case specification from behavior perspective

We introduce ACSExecuteSO action because the user can call system operation in different ways (for example double click on button, pressing specific key on keyboard, focusing on some graphic user component and etc.). At this level, we just emphasize that user just define system operations, but the way how the user does it is only specified at the lowest level. As a result of this level of use case specification, we have identified and specified system function as system operation contracts.

The lower-level of use case specification includes the details about specification user interface. This specification is done in several steps. First, we define appropriate template (e.g. filed-form, table-form, filed-tab, table-tab) which is used to display the main business entity and entities associated with it. Second, from the use case business rule we identify entity and entity attributes that participates in use case and specify the corresponding graphic user interface component used to display and modify its value (e.g. text field, table, dropdown list, radio buttons).Third, for each ACSExecuteSO action we specify the graphic user interface components (e.g. button, menu item) that are used to call system to execute the system operation. Fig.4 suggests the relations between use cases, use case templates, business rules and business entities with corresponding GUI component.

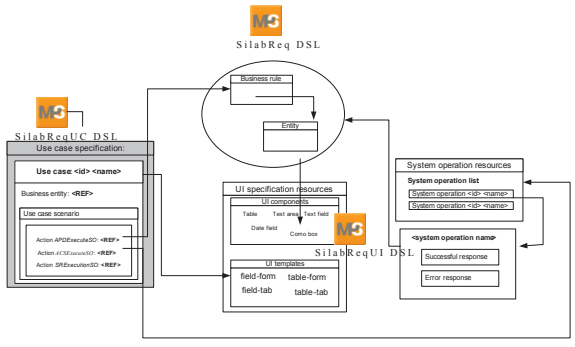

Figure 4 Use case specification from UI perspective

4. THE SILABMDD TOOL SUPPORT

The SilabMDD approach is supported by companion tool, also named SilabMDD tool, that has been developed on top of JetBrains MPS. MPS contains its own language named BaseLanguage. MPS allows extending BaseLanguage to create new custom languages, extend existing languages, and use them to develop software applications. BaseLanguage has a built-in support with strings, collections, regular expressions, etc. During the process of creating a new language it is needed to derive concepts from the BaseLanguage as a reference for new languages.

The major goal of MPS is to allow languages definitions thought extension, which means that language's designer, can use concepts from a new extended language as well as combine concepts from different languages. The problem in syntax language extension is mainly the textual concrete syntax because each language may have its own concrete syntax. JetBrains MPS proposes having concrete syntax maintained in an Abstract Syntax Tree (that consists of nodes with properties, children and references that describes the program code). At the same time, MPS offers an efficient way to keep writing code in a text-like manner.

MPS uses a generative approach that focuses on automating the creation of system-family members: a given system can be automatically generated from a specification written in one or more textual or graphical domain-specific languages [23].

From a developer perspective, this programming approach seems very promising: developers have two ways to implement software. First, they can use requirements specification to manually and formally define their requirements. Second, they can use or create different transformation to generate source code from these models. Both alternatives can be used and integrated with MPS because there is already a plug-in for IntelliJ IDEA which allows including MPS concept models in Java project. So, developers can use MPS for writing Java application and integrate Java source code with SilabReq requirements specifications.

MPS comes with sets of DSL which is use to define the structure of language, editor, type systems, and generators. All of these DSLs are built using MPS itself. The language definition starts by defining its abstract syntax us suggested in Fig.5 (concepts in MPS). The concept is one element of language in MPS which describes how the elements look like, behave and generate[3]. Each concept can have a definition in one or more aspects of language such as structure, editor or generator.

[3] http://dslbook.squarespace.com

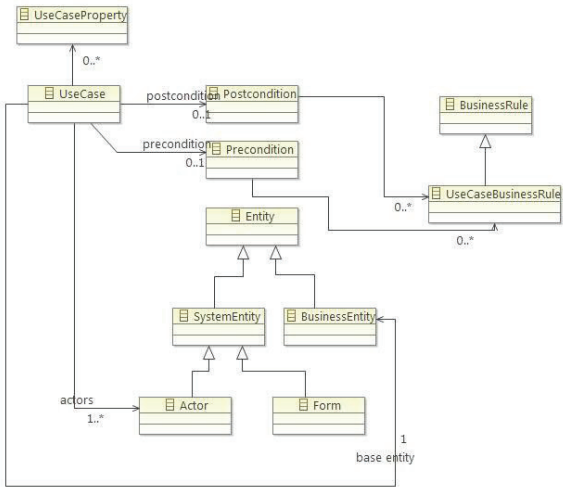

Figure 5 The SilabReq meta-model (partial view)

This part of SilabReqUseCase specification language is described in MPS using its Concept Declaration Editor (Fig.6).

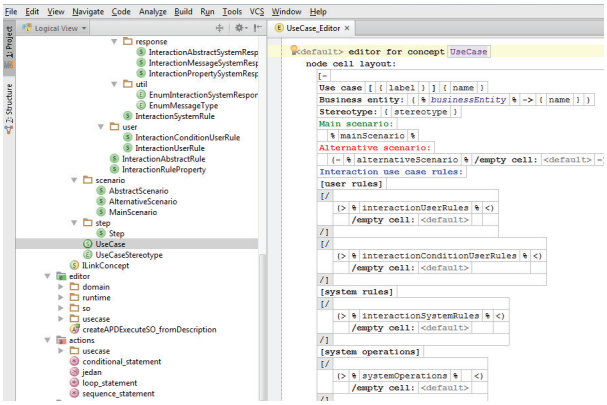

Figure 6 MPS editor for concept declaration

The MPS' Aspect Editor is used for defining the concepts' for concrete syntax.Fig.7 depicts the using of Aspect Editor for UseCase concept.

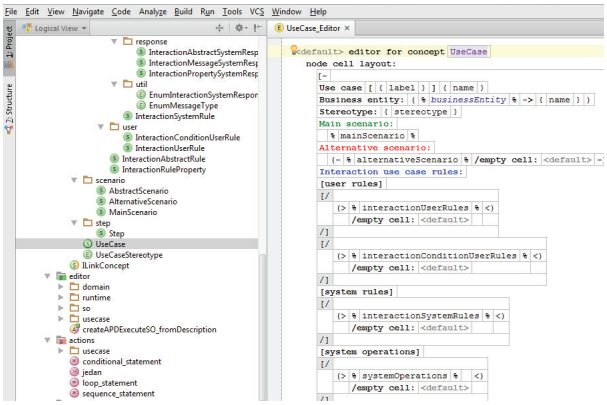

Figure 7 Aspect Editor for Use Case concept

We use MPS to generate Java source code from requirements specification model. Java is embedded into MPS, so generation Java source code is a simple transformation. We use template – based transformation in MPS for generation source code. This transformation has two main important building blocks: mapping rules (define which concepts are processed with which templates), reduction rules (define transformations which removes source node and replace it with associated template) and templates. Fig.8 presents an example of transformation declaration in MPS.

```
mapping labels:
  label inputFieldDeclaration : InputFiled        -> FieldDeclaration
  label useCaseTemplate       : UIUseCaseTemplate -> TemplateGeneration

parameters:
  << ... >>

is applicable:
  <always>

conditional root rules:
  << ... >>

root mapping rules:
  concept          UIUseCase  --> UIUseCase
  inheritors       false
  condition        <always>
  keep input root  default
```

Figure 8 Example of MPS transformation definition

5. CONCLUSION

In this paper we introduce a SilabMDD use-case and model driven approach that includes the SilabReq, a use case specification language. Further, we present how SilabReq is supported by JetBrains Meta Programing System (MPS) framework. SilabMDD is a use case driven approach but it do not pay much attention to the way are elicited use cases. It requires a rigorous definition of the use case specification, particularly description of sequences of action, pre- and post-conditions, and relationships between use cases and business entities.

The goal of SilabMDD is to provide a complete software development workbench to be used by requirements engineers, developers, as well as by other non-technical stakeholders.

In short, the contributions of this article are twofold. Firstly, it introduce SilabReq specification language which can be used for requirements specification. Secondary, it presents SilabMDD as use-case and model driven approach. In this approach use case become the key and central artifact in software development process which are considered at different levels of abstraction.

REFERENCES

[1] IEEE Computer Society Professional Practices Committee SWEBOK®, Guide to the Software Engineering Body of Knowledge. The Institute of Electrical and Electronics Engineers, Inc., 2004

[2] G.Kotonya and I. Sommerville, Requirements Engineering Processes and Techniques. John Wiley and

Sons, 2000Banks, J. and S. J. Carson, Discrete-Event System Simulation, Prentice-Hall, New Jersey, 1984.

[3] IEEE standard glossary of software engineering terminology,IEEE Std610.12-1990, 1990

[4] K.Pohl, Requirements Engineering - Fundamentals, Principles, and Techniques. Springer 2010

[5] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice," Requirements Engineering, vol. 6, no. 11, pp. 4–7, 2004.

[6] S. Mellor, A. Clark and T. Futagami, "Model-Driven Development," IEEE Software, vol. 20, pp. 14-18, 2003.

[7] P. Valderas and V.Pelechano, "A Survey of Requirements Specification in Model-Driven Development of Web Applications", TWEB 5(2):10 (2011)

[8] T.Menzies, ″Editorial: model-based requirements engineering″, Requirements Engi 8(4): 193-194, 2003

[9] G.Loniewski, E. Insfran and S. Abrahão, "A systematic review of the use of requirements engineering techniques in model-driven development", Model driven engineering languages and systems. D. Petriu, N. Rouquette and Ø. Haugen (ed.), Springer: 213-227, 2010

[10] M.Glinz, "Problems and Deficiencies of UML as a Requirements Specification Language", Proc. of the 10th IEEE Int. Workshop on Software Specification and Design, 2000

[11] M. Smiałek, J.Bojarski, W.Nowakowski and T. Straszak, "Scenario construction tool based on extended UML metamodel". Lecture Notes in Computer Science, 3713:414–429, 2005.

[12] M.Smialek and T.Straszak, "Facilitating transition from requirements to code with the ReDSeeDS tool". RE 2012: 321-322

[13] M.Smialek, W.Nowakowski, N. Jarzebowski, A Ambroziewicz," From use cases and their relationships to code" MoDRE 2012: 9-18

[14] A.Silva, C.Videira, J.Saraiva, D.Ferreira and R.Silva, "The ProjectIT-Studio, an integrated environment for the development of information systems", In Proc. of the 2nd Int. Conference of Innovative Views of .NET Technologies (IVNET'06), pages 85–103. Sociedade Brasileira de Computação and Microsoft.

[15] A. R. d. Silva, J. Saraiva, D. Ferreira, R. Silva, and C. Videira, "Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools", IET Software: On the Interplay of .NET and Contemporary Development Techniques, 2007

[16] D. A. Ferreira and A.R.Silva,"A Controlled Natural Language Approach for Integrating Requirements and Model-Driven Engineering", ICSEA 2009: 518-523

[17] D. A. Ferreira and A.R.Silva,"RSLingo: An information extraction approach toward formal requirements specifications", MoDRE 2012: 39-48

[18] Alberto Rodrigues da Silva, João Saraiva, David Ferreira, Rui Silva, Carlos Videira, Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools, in IET Software Journal - Special issue "On the interplay of .NET and contemporary software engineering techniques", December 2007, Volume 1, Issue 6, p. 217-314, IET.

[19] Kostmod4.0 http://rapporter.ffi.no/rapporter/2009/01002.pdf, accessed in January, 2013

[20] F, Martin. Language Workbenches: The Killer-App for Domain Specifc Languagess [online]. Available on: http://martinfowler.com/articles/languageWorkbench.html

[21] D.Savić, A.Rodrigues da Silva, S.Vlajić, S.Lazarević, I.Antović, V. Stanojević, M.Milić, Preliminary experience using JetBrains MPS to implement a requirements specification language, in Proceedings of QUATIC'2014 Conference, 2014 , IEEE Computer Society .

[22] D.Savić, A.Rodrigues da Silva, S.Vlajić, S.Lazarević, I.Antović, V. Stanojević, M.Milić, Use Case Specification at Different Levels of Abstraction, in Proceedings of QUATIC'2012 Conference, 2012 , IEEE Computer Society.

[23] K.Czarnecki, Generative Programming: Methods, Tools, and Applications. Addison-Wesley (2000)