SilabMDD - 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 / Technical University of Lisbon[2]

Abstract - *Model-Driven Development is a software development paradigm that emphasizes the importance of models during the entire software development process. The aim of Model-Driven Development is to use models throughout the software development process at different levels of abstraction. In our SilabMDD approach use cases present a model of user and software requirements. The model of requirements is specified using SilabReqDSL language. SilabReqDSL language is implemented inside JetBrains Meta Programming System and can be used in it as plugin or plugin inside IntelliJ IDEA.*

1. INTRODUCTION

The development of information system 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 is a number of different definitions of requirements. Requirements can be defined as: (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].

The requirements engineering involves two main processes: (1) requirements development (with elicit, analyze, specify, and validate software requirements) and (2) requirements management process [4]. Many requirements engineering approaches have been discussed in the literature, which differ in their analysis 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 10 years later were goal-oriented requirements engineering [5]. Common to all these approaches, particularly in their early period, is the clear separation of requirements engineering process from software development process.

A more software paradigm, referred to as Model-Driven Development (MDD), [6] is a software paradigm that emphasizes the importance of models during the entire software development process. 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 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 requirements engineering 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 requirements engineering level to a code generation level or writing level [8] [9].

In this paper we introduce SilabMDD approach as model drriven, language oriented and use case driven approach that use SilabReqDSL as a use cases specification language. Furthermore, we present how SilabReqDSL 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 other 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, as paragraphs of text. However, natural language requirements specification tends to be ambiguous, unclear, and inconsistent. In fact, they are difficult clearly defined data, function and behavior perspectives of these requirements because they overlap. On the other hand, documenting requirements using semi-formal models require using specific modeling language for each particular perspective. Three types of requirements proposed by Klaus Pohl [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 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 machine diagram or data flow can be used for modeling behavior.

The specification of requirements is a difficult task because requirements are read by many of 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 transformation and also for reusing. We need a structured language for requirements specification that will be understandable by most of participants in software development process but also will be precise enough to enable automatic transformation. These languages should be defined by meta-model or grammars in order to enable automatic or semi-automatic processing.

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

There other Requirements Specification Language (RSL) that use natural language in a controlled way. RSL [11] is a semiformal natural language that employs use case for specifying requirements. Each scenario in 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 ProjejtIT-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 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] .

RSLingo [17] is a linguistic approach for improving the quality of requirements specification, which is based on two languages and mapping between them. The first language is RSL-PL (Pattern Language) extensible language for defining linguistic patterns dealing with information extraction from requirements written in natural language; and the second one is RSL-IL (Intermediate Language), 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 this project 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 based on use cases specifications. When both subprojects reach desired level of maturity, they integrated in a way that some results of SilabReq project can be used as input for SilabUI project. As a proof of concept, Silab project has been used for the Kostmod 4.0 [18] 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

Usually, when we are explaining some approach, we are explaining it according to phase-oriented perspective, defining the roles and tasks involved. In this paper, we do not describe our approach in that manner because we do not focus on phases and roles. We focus on key artifact. In our approach it is use case. We explain it as well as different languages that are used in it.

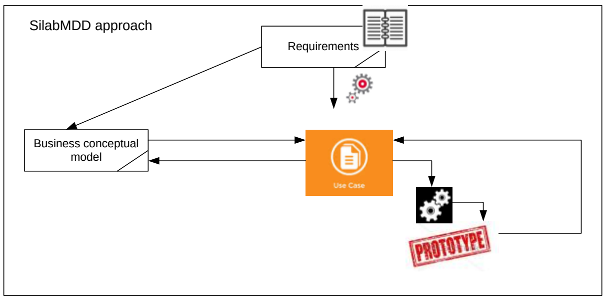


Figure 1Modeling circle in SilabMDD approach

Fig. 1 depicts the key artifacts and modeling circle with SilabMDD approach. SilabMDD approach is a model driven, language-oriented and use case driven approach.

Usually, in MDD the implementation is (semi)automatically generated from the models. Despite the fact that use cases are narratives, there is no a single standard that specify what textual specification of use case should be. In SilabMDD approach we develop SilabReq DSL language that should be used for use case specification. It 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 models and domain models.

Bearing in mind that the model is expressed through a modeling language and SilabMDD approach use several integrated DSL it is also language-oriented. Language Oriented Programming [38] presents a style of development which operates about the idea of building software around a set of DSL, while Language Workbenches are 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 systems that enables language oriented programming with a projection editor in persistent abstract representation [38] that we used in developing our languages.

SilabMDD approach is use case driven approach but it do not pay much attention to the way in which get use cases. It can be derived from business process, or text requirements. If requirements are expressed in some form of model as in RSLingo using with RSL-IL it is possible to automatically using appropriate transformation to deliver use cases. Throughout the specification process use cases are specified using SilabReqUC DSL language and continuous inspection business conceptual model. For business conceptual model description we developed small SilabReqBCM DSL language. Action in use cases as well as pre-condition and post-condition are specified in context of business conceptual model. Except two of this language, SilabMDD approach use SilabReqUI DSL language which is primary use for specification user interface prototype.

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

The SilabReq DSL focuses on use cases specifications at different levels of abstractions according to the different roles that use cases play in the software development. 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). Each abstraction level extends and semantically enriches the previous level. Actually, we can use the same model for both user and system specification and software design. Transformations among these different levels are used to create different views as well as for code generation.

The main role that use cases have at the highest level of abstraction (interaction level) is the 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 connected with glossary and business rules. Glossary and business rules are specified within the same language (but it is possible to create and use specific language). They are specified separately, but they are connected with some part of use cases such as pre-conditions, post-conditions or use case actions (steps). Each business rule or term in the glossary is specified with unique identification, name and description. At this level of abstraction, the use case specification consists in the following elements (see Fig.2): (1) unique use case identifier, (2) use case name, (3) the actors who participate in use case, (4) the business entity over which the use case is executed,(5) main and alternative use case scenarios, and (6) 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 entity 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 entity and login as 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 for use case post-condition specification system. After successful execution of the use case, 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. 1 describes the use case specification template document, entity and business rule specification document. The specification document looks like wiki document; it is possible to navigate through document.

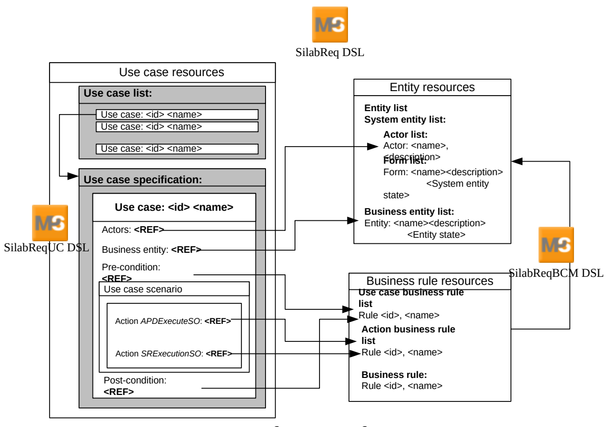


Figure 2 Use case specification from user perspective

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 action and SExecuteSO action. These use case actions are used to define a function that a system should provide. The Fig.3 describes how medium-level use case specification extends high-level. The figure describes template

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

document which is extended with system operation specification that contains: system operation pre-condition, successful and error system response as well as system operation post-condition.

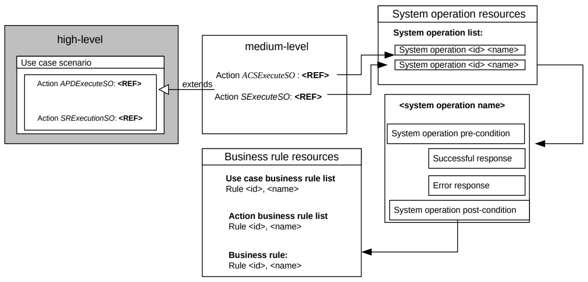

high-level
Use case scenario
Action APDExecuteSO: <REF>
Action SRExecutionSO: <REF>
extends
medium-level
Action ACSExecuteSO : <REF>
Action SExecuteSO: <REF>
System operation resources
System operation list:
System operation <id> <name>
System operation <id> <name>
<system operation name>
System operation pre-condition
Successful response
Error response
System operation post-condition
Business rule resources
Use case business rule list
Rule <id>, <name>
Action business rule list
Rule <id>, <name>
Business rule:
Rule <id>, <name>

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 calls the system operation, but way how the user does it will be 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 contains information about user interface details. The aim of this level of use case specification is to enable specification of user interface.

Specification of use cases at this level of abstraction is done in several steps. First, we define appropriate template (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. The Fig.4 describes relation between use cases and appropriate use case templates, business rules and business entities with corresponding graphic user interface component.

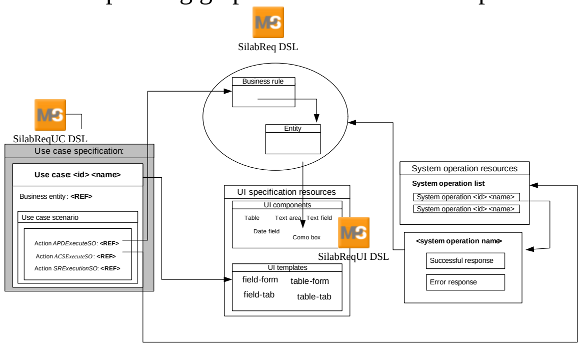


Figure 4 Use case specification from UI perspective

C. The supported tool: 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 concept 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 of 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 [20] , similarly to MDD that aims to capture important aspects of a software systems through appropriate models.

From developer perspective, this programming approach seems very promising. Developer has two ways to implement software. First, he can use requirements specification to manually create source code. Second, he 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, programmer can use MPS for writing Java application and integrate relationship between source code and requirements specification.

D. Main aspect of SilabReq implementation

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

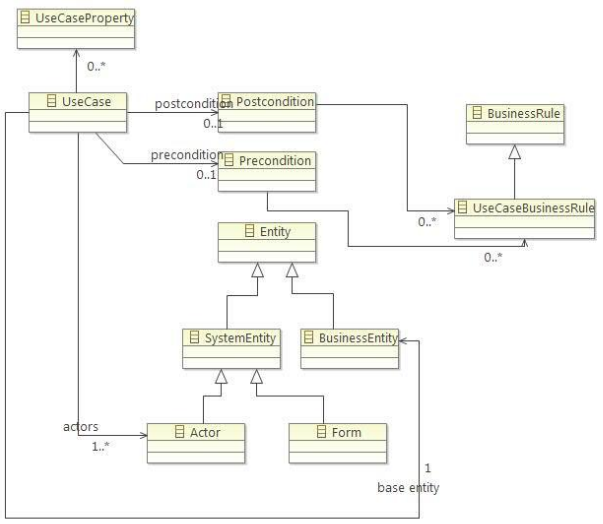
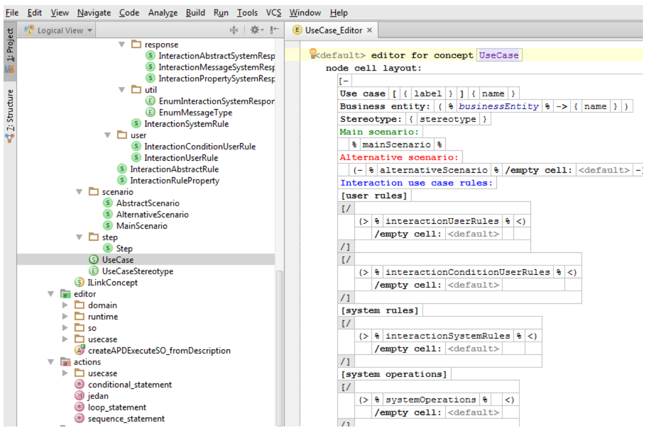


Figure 5 The SilabReq meta-model (partial view)

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

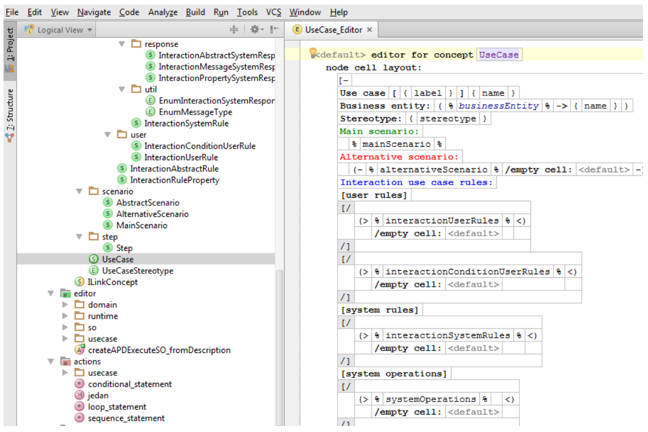

File Edit View Navigate Code Analyze Build Run Tools VCS Window Help
1: Project
7: Structure
Logical View
UseCase_Editor

```
▼ response
    InteractionAbstractSystemResp
    InteractionMessageSystemResp
    InteractionPropertySystemResp
▼ util
    EnumInteractionSystemRespor
    EnumMessageType
  InteractionSystemRule
▼ user
  InteractionConditionUserRule
  InteractionUserRule
InteractionAbstractRule
InteractionRuleProperty
▼ scenario
  AbstractScenario
  AlternativeScenario
  MainScenario
▼ step
  Step
UseCase
UseCaseStereotype
ILinkConcept
▼ editor
  ▶ domain
  ▶ runtime
  ▶ so
  ▶ usecase
  createAPDExecuteSO_fromDescription
▼ actions
  ▶ usecase
  conditional_statement
  jedan
  loop_statement
  sequence_statement
```

```
<default> editor for concept UseCase
  node cell layout:
    [-
    Use case [ { label } ] { name }
    Business entity: ( % businessEntity % -> { name } )
    Stereotype: { stereotype }
    Main scenario:
      % mainScenario %
    Alternative scenario:
      (- % alternativeScenario % /empty cell: <default> -)
    Interaction use case rules:
    [user rules]
    [/
      (> % interactionUserRules % <)
        /empty cell: <default>
    /]
    [/
      (> % interactionConditionUserRules % <)
        /empty cell: <default>
    /]
    [system rules]
    [/
      (> % interactionSystemRules % <)
        /empty cell: <default>
    /]
    [system operations]
    [/
      (> % systemOperations % <)
        /empty cell: <default>
    /]
```

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.

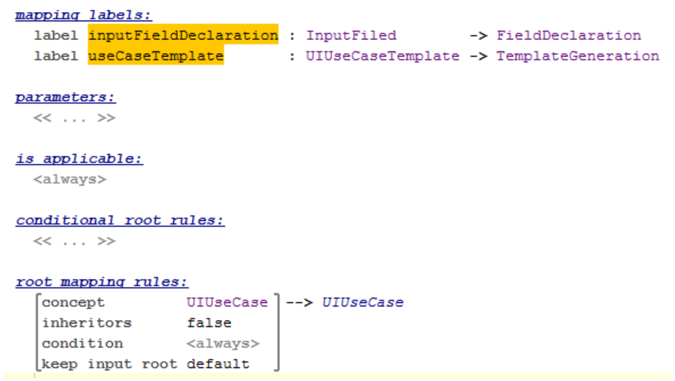


```
▼ response
    InteractionAbstractSystemResp
    InteractionMessageSystemResp
    InteractionPropertySystemResp
▼ util
    EnumInteractionSystemRespor
    EnumMessageType
  InteractionSystemRule
▼ user
  InteractionConditionUserRule
  InteractionUserRule
InteractionAbstractRule
InteractionRuleProperty
▼ scenario
  AbstractScenario
  AlternativeScenario
  MainScenario
▼ step
  Step
UseCase
UseCaseStereotype
ILinkConcept
▼ editor
  ▶ domain
  ▶ runtime
  ▶ so
  ▶ usecase
  createAPDExecuteSO_fromDescription
▼ actions
  ▶ usecase
  conditional_statement
  jedan
  loop_statement
  sequence_statement
```

```
<default> editor for concept UseCase
  node cell layout:
    [-
    Use case [ { label } ] { name }
    Business entity: ( % businessEntity % -> { name } )
    Stereotype: { stereotype }
    Main scenario:
      % mainScenario %
    Alternative scenario:
      (- % alternativeScenario % /empty cell: <default> -)
    Interaction use case rules:
    [user rules]
    [/
      (> % interactionUserRules % <)
        /empty cell: <default>
    /]
    [/
      (> % interactionConditionUserRules % <)
        /empty cell: <default>
    /]
    [system rules]
    [/
      (> % interactionSystemRules % <)
        /empty cell: <default>
    /]
    [system operations]
    [/
      (> % systemOperations % <)
        /empty cell: <default>
    /]
```

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

4. CONCLUSION

In this paper we introduce SilabMDD approach as model drriven, language oriented and use case driven approach that use SilabReqDSL as a use cases specification language. Further, we present how SilabReqDSL is supported by JetBrains Meta Programing System (MPS).

SilabMDD approach is use case driven approach but it do not pay much attention to the way in which get use cases. It 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 models and domain models.

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: (1) the introduction of SilabReq specification language which can be used for requirements specification, and (2) presend SilabMDD approach as model drriven, language oriented and use case driven approach. Achieving this goal, in our apparoach use case becomes the main artifact in software development which is considered at different level 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] Kostmod4.0 http://rapporter.ffi.no/rapporter/2009/01002.pdf, accessed in January, 2013

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

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