2012 Eighth International Conference on the Quality of Information and Communications Technology

Use Case Specification at Different Levels of Abstraction

Dušan Savić
Software engineering Laboratory
Faculty of Organizational Sciences
Belgrade, Serbia
dules@fon.bg.ac.rs

Alberto Rodrigues da Silva
Department of Computer Science and Engineering
IST / Technical University of Lisbon
Lisboa, PORTUGAL
alberto.silva@acm.org

Siniša Vlajić, Saša Lazarević, Vojislav Stanojević, Ilija Antović, Miloš Milić
Software engineering Laboratory
Faculty of Organizational Sciences
{vlajic, slazar, vojkans, ilijaa , mmilic }@fon.bg.ac.rs

***Abstract*—Use cases are narrative description of interactions between users and a software system. They are primary used to specify functional system requirements. Despite the fact that use cases are narrative, there is no standard that specifies what textual specification of use case should like. However, the integration of use cases in a Model Driven Engineering process requires more rigorous specifications. The specification of use cases at different levels of abstraction is needed because of the large number of stakeholders involved in the software development process. We propose use cases specification at different levels of abstraction to promote better integration, communication and understanding among the involved stakeholders. We still discuss that base on this approach different software development artifacts such as domain model, system operations, user interface design can be automatically generated or validated.**

Keywords: use case; use case specification; use case meta-model

I. INTRODUCTION

The software requirements engineering process is a part of software development process and one of the key processes in software development. The elicitation, analysis, specification and validation of software requirements occur during the requirements engineering process. The requirements can be presented in different ways. Herman emphasizes two forms of requirements representation [1]: descriptive requirements presentation and model-based presentation of the requirements.

The result of the requirement engineering process should be clear and precise specification of system functions that need to be implemented. Use cases are used as a technique for functional specification [2, 3, 4, 5]. Use cases describe interactions between actors and the system in the consideration. Therefore, when we define use cases it is necessary to define: (1) the actors and (2) the interactions between an actors and the system. The UML specification defines a use case as a sequence of actions, but the specification of these actions is not clearly defined, and so different notations can be used to describe these actions. UML uses different types of diagrams (sequence diagram, activity diagram, and communication diagram) to describe the interactions within a use case. On the other hand, textual descriptions of use cases have also been proposed in the work of Rolland's [6], Cockburn [2], and Li [7]. Furthermore, formalisms based on Petri nets [8] and formal specification languages, such as Z[4] have been used for use cases specification. The main blemish of formal notation is that they are very difficult to understand by individual participants in a software project. The integration of use cases within Model Driven Engineering (MDE) [9] requires a better definition of the use case specification, particularly description of sequences of action steps, pre- and post- conditions, and relationships between use case models and conceptual models [10].

In this paper we propose idea of the use cases specification at different levels of abstraction using SilabReq domain specific language [11] which is a part of Silab project. We focus our interest on notation that allows a human reader with no or little formal background to quickly read and understand use case descriptions.

Use cases contains one main scenario and zero or more alternative scenarios while each scenario contains one or more use case actions. There are two categories of actions: (1) actions performed by users; and (2) actions performed by the system.

In the category of actions performed by the user, there are the following actions types: (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 of actions performed by the system, there are two action types: (2.1) System executes System Operation (*SExecuteSO*); and (2.2) System replies and returns the Result of the System Operation execution (*SRExecutionSO*). Different types of action are specified at different use case levels.

These actions appear at the appropriate level of abstraction and are specified in requirements specification document using SilabReq DSL or/and UML. SilabReq DSL has been developed using Xtext framework [12]. By defining use case actions in a clear and precise manner,

978-0-7695-4777-0/12 $26.00 © 2012 IEEE
DOI 10.1109/QUATIC.2012.64

the use case model is described more rigorously. Additional, by applying different transformations on the use case model, it is possible to get different models that can describe system boundary, the structure of the system, or the system behavior as a set of function that the system should provide.

This paper is organized as follows. In the Section II, we put this proposal in the context of some related work. Section III gives an overview of the Silab project. Section IV presents use case at different levels of abstraction with a concrete example. Finally, Section V concludes the paper and outlines future work.

II. RELATED WORK

Different forms or templates to write use case specification for functional requirements, defined by different authors, usually contain parts that can be found in almost all templates. In practice, the most commonly used templates are: Cockburn's use case template [2] and Rational Unified Process' use case template [4].

Although there is no standard that specifies what textual representation of different use case behavior should like, some authors have proposed templates or metamodel for this purpose. Siqueira et al. analyzes these studies and present a metamodel based on an analysis of 20 studies [13]. However, this metamodel only considers the most frequent concepts such as use case, actor, main and alternative scenario, condition, action and does not consider different type of these actions.

In [14, 15] Somé analyzes two templates to create UML-compliant metamodel, focusing on consistence with the UML specification and defines the abstract syntax of a textual presentation of the use case. The author emphasizes that certain elements of the UML such as actions and activities are formally defined through meta-model, while on the other hand, UML has not formally defined meta-model for use cases textual description, although text-based notation is a primary notation for the use case description. Therefore, Somé defines a meta-model to describe interactions between users and systems.

In [16] Smialek suggests a different notation for the use cases description. He suggests that we need to have different notation for different user and he related certain notation of the use case with user's role in the software development process. He emphasizes that the ideal notation for use case description should be enriched because different users should have different views for the some use case, namely: User's, Analyst's, Designer's, User interface designer's and Tester's point of views.

Therefore, Smialek proposes four different notations for the use cases description that are based on structured text, interaction diagrams and activity diagrams. In the same paper, he defines a meta-model for structured textual presentation of use case which is based on simple grammatical sentences. These sentences are in the form of "subject-verb-direct object", but can also appear in the form of "subject-verb-indirect object".

Ivar Jacobson has introduced different type of these actions such as: the primary actor sends a request and data to the system, the system validates the request and the data, the system alters its internal state, and the system replies to the actor with the result. [17].

Similarly, Williams et al. defined four types of actions (Input, Output, Computation, and Exception Handling) and four flows (Selection, Iteration, Inclusion and Extension) within these actions are executed [18].

Further, in [19] Genilloud at al. discussed limitations of the graphical notation for use cases specification and emphasized that use cases specification in natural language should describe the actors' actions of entering data, selecting an option, etc., so that they can be easily read by end users and other stakeholders. Apart from identification of these use cases actions, contrary to our proposal, they have not suggested how to specify these actions. Our approach proposes how to specify these actions. Cockburn suggest that these actions (steps) should be written as "subject + verb + direct object + preposition phase" while Smialek [16] proposes different patterns and grammar to describe them.

Our approach is closely related with ProjectIT [20, 21, 21, 23] in the following: (1) both approaches use controlled natural language for requirements specification, (2) support requirements-to-models, models-to-models, and models-to-code transformation techniques; and (3) integrate requirements engineering with MDE. Contrary to ProjectIT approaches [21] that identified a common set of linguistic patterns for the requirements of interactive systems and derived a corresponding metamodel of requirements that adequately represents the patterns identified, our proposal use controlled natural language for use case specification. Additionally, we intend to integrate our proposal with ProjectIT to improve the software development process as a whole.

The main differences between our approach and others that define a specific format to describe actions, lies in the fact that we focus on use case execution in the context of a domain model over which the use case is performed. A domain model is a model of the significant entities from the system's application domain. These are the entities that will be created, modified, used, and links between them may be added or deleted. Domain model are represented typically as UML class diagrams.

We have identified several types of actions that appear in the execution of use case scenarios and divided it in two categories: the actions performed by the user; and the actions performed by the system. The SilabReq domain specific language developed for use case specification, on one hand allows specification of these actions, so that they can be readable and understandable to all participants in the projects and, on the other hand, contains semantics so that it is possible to discover the structure of the system (the domain model of the system) and the behavior of the system (the functions that the system should provide). User interface (UI) details must be kept out of the functional specification, but needs to be linked to it. We propose link UI details with use cases, but only specified

these details in lower abstraction level. Section IV gives overview how we discover domain model, system operation and ability to automatically create an executable prototype of the system from use case specification.

III. SILAB PROJECT

Silab Project was initiated in Software Engineering Laboratory at Faculty of Organizational Sciences, University of Belgrade, in 2007. 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.

Initially this project has been divided in two main subprojects SilabReq and SilabUI that were being developing separately. SilabReq project considered formalization of user requirements and transformations to different UML models in order to facilitate the analyses process and to assure the validity and consistency of software requirements. SilabReq language is the main result of this project. On the other hand, SilabUI project considered automatic generation of user interface based on the use case specification.

When both subprojects reach desired level of maturity, they were 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 [24] project, which was implemented for the needs of the Royal Norwegian Ministry of Defense.

The SilabReq project includes SilabReq Language, SilabReq Transformation and SilabReq Visualization components.

A. Silab language

The SilabReq language component is controlled natural language for use case specification. Our proposal suggests writing use case specification, like a word processor, and as we write, it will warn us of errors violating the requirements language and grammar rules we have defined. It is developed under XText framework [12]. This framework is based on openArchitectureWare generator [25] framework, the Eclipse Modeling Framework (Eclipse Modeling Framework) [26] and Another Tool for Language Recognition (ANTLR) parser generator [27]. The framework uses the BNF grammar for the description of concrete syntax of the language. On the other hand, based on the BNF grammar, the framework creates a meta-model that describes the abstract syntax of the language.

B. Silab transformation

The SilabReq transformation component is responsible for transformation requirements specified in SilabReq language into different models. Currently, we have developed transformations that transform SilabReq model into UML models: (1) SilabReq2DomainModel transformation generates domain model, (2) SilabReq2SystemOperation transformation generates system operations, (3) SilabReq2UseCaseModel transformation generates UML use case model, (4) SilabReq2SequenceModel generates UML sequence model, (5) SilabReq2StateMachineModel generates UML state-machine model, and (6) SilabReq2ActivityModel generates UML activity model. The SilabReq transformations are defined through Kermeta language for meta-modeling [28]. Kermeta is a model-oriented language fully compatible with OMG Essential Meta-Object Facility (EMOF) meta-model and Ecore meta-model, which is part of the Eclipse Modeling Framework (Eclipse Modeling Framework EMF) [26].

C. Silab visualisation tool

The SilabReq visualization component is responsible for visual presentation of the specified software requirements. Currently, we have developed only UML presentation of these requirements through UML use case, UML sequence, UML activity or even UML state-machine diagram.

IV. USE CASE ABSTRACTION LEVELS

Use case driven development considers use cases as the basis for the software development [29]. While they have been successful in having an impact on software development, their usage is not as prevalent as we believe it could be [28]. Williams at el. point out several factors for this cause: (1) use cases are captured in many different notations, (2) use case semantics is poorly defined, and (3) misaligned characterizations of use cases in UML meta-model. In our approach, in initial stages of the software development, use cases are primary used by customers, end users, requirements engineers, domain experts and business analysts. But, in later stages use cases are also used by designers, software developer, programmers, user interface designers and testers.

When considering use cases at different levels of abstraction we discussed issues such as: (1) What can we obtain from use cases? (2) Which roles do they have in software development? (3) Do we use them only to present interaction between users and system, or do we use them to create or identify domain model or do we use them to elicit functional requirements of the system.

We consider three different levels of abstraction according to the role that use cases play in the software development: (1) interaction use case specification level (high-level), (2) behavior use case specification level (medium-level), and (3) UI based use case specification level (lower-level). Each abstraction level extends and semantically enriches the previous level.

A. Simple Example

This section presents three different levels of use case abstraction as well as a simple order system. It is a simple application, but it shows all the major relevant aspects of our proposal. The specification of order system starts from the high level specification. Before actor register new offer, he need to: (1) enter the basis information

about the offer, (2) find and select an appropriate business partner for offer, (3) enter offer details, and (4) call system to save new offer in the system.

B. High-level Use case specfication

The main role that use cases have at the highest level of abstraction (interaction level) is the specification of user system interaction. In our approach, use cases at this level are specified with SilabReq language. SilbReq is a textual domain specific modeling language. The abstract syntax of SilabReq DSL is defined using SilabReq meta-model, while the concrete textual syntax is defined using a grammar. The Fig..1. shows the part of SilabReq use case meta-model at this level of abstraction. The grammar of SilabReq DSL is specified under XText framework and it is out of the scope of this paper; for detail see [11].

At this level the use case specification involves the specification of: (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, and (5) main and alternative use case scenarios.

SilabReq metamodel is used to define the constructs in SilbReq DSL. Therefore, the constructions such as *Actor, BusinessEntity Directive*, *Iterate* are defined using appropriate metaclasses defined in SilabReq metamodel.

The main use case scenario (*MainUseCaseScenario metaclass)* consists of one or more use case blocks (*CompleteActionBlock metaclass*). Each use case block contains: (1) the action block which executes the actors (*UserActionFlow metaclass*) and, (2) the action block which executes the system (*SystemActionFlow metaclass*). At this abstraction level, the action block that executes actors contains only zero or more APDExecuteSO (*APDExecuteSO metaclass*). Action block that executes the system, at this level contains only one action *SRExecutionSO*.

During the specification of *APDExecuteSO* actions we need to specify the data that actor enters. Each *APDExecuteSO* actions is associated with an appropriate business rule (*BusinessRuleActorEnters metaclass*) that describe data that user enters/chooses. The actor enters or chooses a business entity (*BusinessEntity metaclass*) and/or the attributes of business entity (*BusinessEntityProperty metaclass*).

Each action performed by an actor is performed in some control structures, namely: (1) sequence control structure (*SequenceUserActionFlow metaclass*), (2) repetition control structure (*IterateUserActionFlow metaclass*), or (3) selection control structure (*ChooseUserActionFlow metaclass*). These structures can contain include or/and extends directive (*DirectiveAction metaclass*) to make association between use cases.

During the specification of the *SRExecutionSO* action we need to specify system response that can be some message or some data. The Fig.2 presents the specification of order system at interaction level.

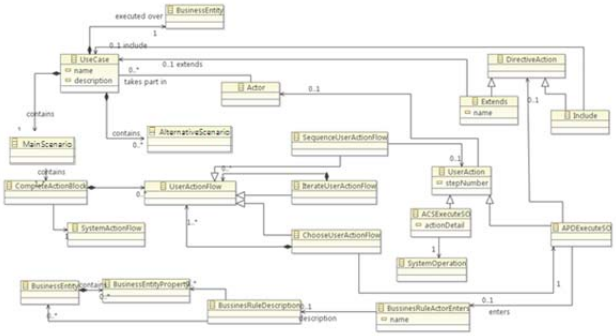

Figure 1. SilabReq language meta-model

Figure 2.Use case interaction level

C. Medium-level use case specification

The specification of use cases at the medium level (behavior level) is used by domain experts, requirements engineer, business analyst, as well as software developers to determine the desired functionality of the system and determine appropriate domain model. Therefore, the use case specification at this level of abstraction includes: (1) extension of the use case specification from the previous level of abstraction, (2) specification of domain model, and (3) specification of the system operations that the system should provide. SilabReq DSL and UML are used at this level of abstraction. Therefore, use case specification extension is made through SilabReq DSL and includes: (1) the specification of the use case pre-condition and post-condition (2) the specification of the use case priority, (3) a detailed specification of the main and alternative use case scenarios. The actions of the main use case scenario performed by the actors are extended with *ACSExecuteSO* actions, while the actions performed by the system are extended with *SExecuteSO* actions. At this level of abstraction, *SRExecutionSO* actions are not showed because the result of system operation execution is specified in the specification of the system operation itself.

At this level of abstraction, each block of actions performed by the user consists of the *APDExecuteSO* actions (if any), as defined in the previous level of

abstraction, and specification of an *ACSExecuteSO* action. *ACSExecuteSO* action uses actor to send request to the system to execute the system operation. The specification of the system operation contains information about: (1) system operation pre-condition, (2) the system operation name and, (3) result that system operation return to the actor. Each use case at this level of abstraction contains a section for defining the system operations. Each system operation defines a pre-condition, post-condition, and successful and unsuccessful result of system operation execution.

The Fig. 3. schematically shows transformation from high-level to medium-level use case abstraction level. The use cases at the high level of abstraction are specified using SilabReq DSL. *SilabReqConceptModel* transformation reads this use case specification and generates domain model for that use case. Transformation finds the primary business domain object over which the current use case is executed and, creates the associations with the other business objects that are found in use case specification. These business objects are observed by the inspection of appropriate business rules. In our example, this transformation primarily creates *Offer* domain business class, and later during inspection of business rules (*BRule_3*, *BRule_4*) creates *OfferDetails* business class and associates these business classes. Because the business rule *BRule_4* is performed iteratively, our transformation creates UML aggregation relationship (0..*) between these classes. Except business classes this transformation also recognizes business class attributes.

At this level of use case abstraction, except *SilabReqConceptModel* transformation we have proposed *SilabReqSystemOperation* transformation that inspect use case specification and generate system operations that system should provide. The system operations are discovered by inspecting *ACSExecuteSO SExecuteSO* and actions. Based on the specification of the *APDExecuteSO* actions the transformation identifies input parameters, while using the specification of the post-condition transformation generates system operation return type.

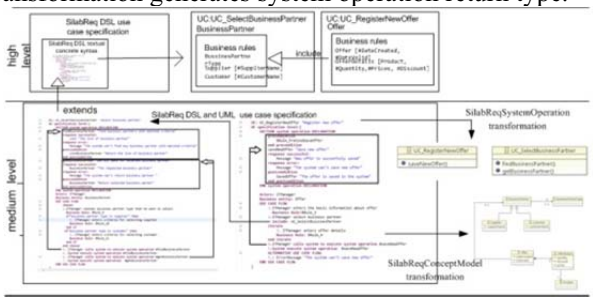

Figure 3. From high to medium-level use case specification

D. Low-level use case specification

In our approach, UI based specification level is low-level of abstraction. This level contains information about user interface details and it can be used for developing user interface prototype as well as the complete user interface.

Specification of use cases at this level of abstraction is done in several steps. First, we need to define the appropriate template which is used to display the main business object and objects associated with them. Second, for each attribute of domain object we need to specify the corresponding graphic user interface component used to display and modify its value (e.g. text field, table, drop-down list, radio buttons).Third, in the end, we need to define the graphic user interface components (e.g. button, menu item) used to call system to execute system operation. A single use case can be realized through different templates, namely:

(1) The field-form template was devised in such a way as to enable the input and output of information related to a specific entity through the use of fields. Each attribute of specific entity is displayed in the appropriate GUI components. Details about related entities are displayed in separate windows by pressing the appropriate option button. Each related entity can be displayed using a different template.

(2) The field-tab template, much like the field-form template, was devised in such a way as to enable the input and output of information related to a specific entity through the use of fields while the entities related with the specific entity are displayed in different tabs. Each related entity can be displayed through a different template.

(3) The table-form template is devised in such a way to enable the input and output of information about a specific entity in the form of a table. Details about related entities are displayed in separate windows by pressing an appropriate option button. Each related entity can be displayed using a different template.

(4) The table-tab template, like the table-form template, is devised to allow the input and output of information about a certain entity in a table form, while the related entities are displayed in the same window, but in different tabs. Each related entity can be displayed using a different template.

The Fig. 4. shows use case specification at the UI-based level. In our example, use case "*Register new offer"* is annotated as a field-tab template. The attributes of the *Offer* entity are presented as text fields (date created, date valid), while the related entities (*BusinessPartner* and *OfferDetails*) are presented in separate tabs. *OfferDetails* entity is specified as a table-form, while *BusinessPartner* entity is specified as a field-form template.

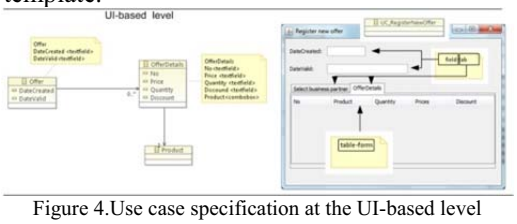

Figure 4.Use case specification at the UI-based level

V. CONCLUSION

Use cases are "narrative" approaches to requirements engineering because they describe the context and requirements in natural language. Therefore, use cases are primarily textual representation techniques. Each use case consists of one or more scenarios that describe how the system should interact with the user or other systems to achieve a particular goal. UML uses different types of diagram (e.g. sequence, activity or state machine diagrams) to help specify these actions. The key point in describing use cases lies in their readability and understanding for the majority of participants in a software project. Since there are many participants involved in software development process with different technical knowledge, the specification of use cases at different levels of abstraction is needed. In this paper we have introduced an approach where use cases can be specified at three different level of abstraction in an integrated way preserving the traceability between these levels and based on MDE-based transformation techniques.

For future work, besides improving, we will work on an appropriate tool which will support this approach as a whole. Additionally, we intend to apply this approach in controlled experiment in order to refine and validate it.

References

[1] K.Herman and D.Svetinovic, "On confusion between requirements and their representation", Requirements Engineering, Springer-Verlang, 2010.

[2] A.Cockburn, Writing Effective Use Cases. Addison-Wesley, New York, 2000.

[3] M.Glinz, "Problems and Deficiencies of UML as a Requirements Specification Language", In IWSSD '00: Proceedings of the 10th International Workshop on Software Specification and Design, pages 11-22, Washington, DC, USA, IEEE Computer Society,2000.

[4] J.Spivey, The Z Notation: A Reference Manual. Prentice Hall, 1992.

[5] I.Jacobson, G. Booch and J.Rumbaugh, The Unified Software Development Process, Addison-Wesley, New York, 1998.

[6] C.Rolland and C.B. Achour, "Guiding the construction of textual use case specifications", In Data & Knowledge Engineering, volume 25 no. 1-2, pages 125-160, 1998.

[7] L.Li, "Translating use cases to sequence diagrams", In Proc. of Fifteenth IEEE International Conference on Automated Software Engineering, pages 293-296, Grenoble, France, 2000

[8] J.B.Jorgensen and C.Bossen, "Executable Use Cases: Requirements for a Pervasive Health Care System". IEEE Software, vol 21(2):34-41, 2004.

[9] G. Loniewski, E. Insfran, and S. Abrahao, "A Systematic Review of the Use of Requirements Engineering Techniques in Model-Driven Development", In: Petriu, D.C., Rouquette, N., Haugen, (eds.) MoDELS. Lecture Notes in Computer Science, vol. 6395, pp. 213-227, Springer (2010)

[10] H. Astudillo, G. Génova, M. Smialek, J. L. Morillo, P. Metz, and R. Prieto-Díaz, "Use cases in Model-Driven Software Engineering", In Proceedings of MoDELS Satellite Events,2005. pp.272-279

[11] D.Savic, I.Antovic, V.Stanojevic, M.Milic and S.Vlajic, "Language for Use Case Specification" In: Proceedings of the 34th annual IEEE Software Engineering Workshop (SEW34). IEEE Computer Society, 2011.

[12] Xtext http://www.eclipse.org/Xtext/documentation/,accessed in May, 2012

[13] F.L.Siqueira and P.S.Muniz Silva, "An Essential Textual Use Case Meta-model Based on an Analysis of Existing proposals", WER 2011.

[14] S.Som´e, "Supporting Use Cases based Requirements Engineering", Information and Software Technology, 48(1):43–58, 2006

[15] S. Somé, "A Meta-model for Textaul Use Case Description", Journal of Object Techology, Zurich, 2009

[16] M.Smialek, **"**Accommodating Informality with Necessary Precision in Use Case Scenarios", Journal of Object Technology,2005.

[17] I.Jacobson, Object-Oriented Software Engineering: A Use-Case Driven Approach, Addison-Wesley, Reading, MA, 1992.

[18] C.E Williams, "Towards a test-ready meta-model for use cases", pUML'2001. pp.270~287

[19] G. Genilloud , W. F. Frank , and G. Génova, "Use cases, actions, and roles", Satellite Events at the MoDELS 2005 Conference, Lecture Notes in Computer Science, Springer Berlin, 2006

[20] A.Silva, J.Saraiva, R.Silva and C.Martins, "XIS – UML profile for eXtreme modeling interactive systems", Proc. 4th Int. Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES '07) (Los Alamitos, CA: IEEE Computer Society), Braga, Portugal, March 2007, pp. 55–66

[21] C.Videira, D.Ferreira and A.Silva, "A linguistic patterns approach for requirements specification", Proc. 32nd EUROMICRO Conf. Software Engineering and Advanced Applications (EUROMICRO'06) (Washington, DC: IEEE Computer Society), Dubrovnik, Croatia, August 2006, pp. 302–309

[22] 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.

[23] 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.

[24] Kostmod4.0 http://rapporter.ffi.no/rapporter/2009/01002.pdf, accessed in May, 2012

[25] openArchitectureWare, http://www.openarchitectureware.org, accessed in May, 2012

[26] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/ accessed in May, 2012

[27] ANother Tool for Language Recognition, http://www.antlr.org/, accessed in May, 2012

[28] Kermeta, http://kermeta.org/, accessed in May, 2012

[29] I.Jacobson , M. Christerson , P. Jonsson , G.Overgaard Object-Oriented Software Engineering –A Use Case Driven approach, Addison Wesley Longman Publishing Co. Inc, Boston, 1993