DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2014-0008

2014/16

Introduction to the Integrated Domain Modeling Toolset

Armands Slihte[1], Juan Manuel Cueva Lovelle[2],
[1]*Riga Technical University, Latvia*, [2]*University of Oviedo, Spain*

***Abstract* – This paper describes the Integrated Domain Modeling approach and introduces the supporting toolset as a solution to the complex domain-modeling task. This approach integrates artificial intelligence (AI) and system analysis by exploiting ontology, natural language processing (NLP), use cases and model-driven architecture (MDA) for knowledge engineering and domain modeling. The IDM toolset provides the opportunity to automatically generate the initial AS-IS model from the formally defined domain knowledge. In this paper, we describe in detail the scope, architecture and implementation of the toolset.**

***Keywords* – Eclipse modeling framework, model-to-model transformation, natural language processing, topological functioning model.**

I. Introduction

In the context of software engineering, a domain is most often understood as an application area, a field for which software systems are developed [1]. An accurate domain model can also be used as input to solution implementation within a software development process since the model elements comprising the problem domain can serve as key inputs to code construction, where construction is achieved manually or through automated code generation approaches. Domain model is an integrated system of models that reflect the enterprise where software is to be applied [2]. In other words, a domain model is the AS-IS model of the business organization and processes. Domain modeling is a human activity that leads to the creation of different types of domain representations. These representations may be tacit (in human minds) and explicit/externalized (on paper or in a software tool) [3]. Domain analysis can be seen as a process where information used in developing software systems is identified, captured, structured, and organized for further reuse [1]. This paper discusses the Integrated Domain Modeling (IDM) approach for the domain analysis and introduces the supporting toolset for this approach developed by the author. This toolset is based on Eclipse [4] and consists of several tools that interoperate to provide a possibility for a particular domain model construction.

The ultimate goal of this research is to lower the cost of software development by introducing a methodology and a toolset, which would allow a comprehensive analysis of the domain at the beginning of software development and, thus, minimizing the count of bugs and necessary corrections due to an inconsistent understanding of the domain. The main contribution of this paper is the introduction of the IDM supporting toolset, discussing the architecture, used technology, model transformation and the use of this tool to acquire a domain model.

There are several approaches to domain modeling, which have also a supporting toolset, but usually they require the users to learn a new form of domain knowledge accepted by the approach and the tools and then manually construct the domain model. The IDM approach, on the other hand, integrates the common standards used in business environment – use cases and ontology, and then provides the means to generate the initial domain model automatically. This level of automation and reuse of enterprise standards is the main innovation of the IDM approach. The IDM provides the opportunity to do this by using the Topological Functioning Model (TFM) as the domain model. Thus, the construction of the domain model is made simpler by exploiting model transformations, natural language processing (NLP), use cases and ontology.

II. Related Work

There are other approaches, which suggest constructing the domain model based on domain knowledge. The authors give a short overview of some of these approaches in this section. Use Cases are defined with a natural language, so Natural Language Processing (NLP) can be used for an analysis. Approach discussed in [5], called NIBA (natural language requirements analysis in German), addresses the same issues. By following the "NIBA workflow", natural language requirements specifications are translated into conceptual predesign schema. After validation by the user, the predesign schema is mapped to a conceptual representation (e.g., UML). Approach proposed in [6] suggests generating implementation from textual use cases. This approach uses statistical parser on use cases and by analyzing the parse trees composes the so-called Procases for further use in implementation generation. Another approach ReDSeeDs [7] defines software cases to support reuse of software development artifacts and code in a model driven development context. This approach is very formal and it depends on writing the software cases very precisely by adding specific meaning to every word or phrase of software case sentences (the purpose is similar to use case and use case steps). The Use Case Driven Development Assistant (UCDA) tool methodology follows the IBM Rational Unified Process (RUP) approach to automate the class model generation [8]. First, the requirements of the system are analyzed identifying the use cases and actors of the system. Using these artifacts the tool can generate the UML use case diagram, class diagram, communication diagram, and

other artifacts. This tool utilizes natural language processing methods for processing the requirements in textual form.

III. The Integrated Domain Modeling Approach

This paper is part of the Topological Functioning Model for Software Engineering (TFM4SE) research. TFM is a domain model, which offers a formal way to define a system by describing both the system's functional and topological features. Related research suggests using TFM as a Computation Independent Model (CIM) by constructing it with a Topological Functioning Model for Model Driven Architecture (TFM4MDA) approach [9]–[13], acquiring a mathematically formal and thus transformable CIM. In related research [18], the TopUML approach is described for software development with emphasis on topology, where Platform Independent Model/Platform Specific Model (PIM/PSM) is supplemented with topology. TopUML is the UML profile and approach to introducing cause and effect relationships into the UML based on the topology of TFM. TopUML approach suggests sequential phases of TFM4MDA approach to be combined for fulfilling the Model Drive Architecture (MDA) life cycle taking the TFM as a source for PIM/PSM.

Although the TFM, TFM4MDA and TopUML provide a solid basis for CIM construction within MDA and further transformations to PIM/PSM, until now the construction of the TFM relies on a heavy manual process with no tool support and a poor integration with the common IT practices. The AS-IS processes of TFM4MDA are described in [14]–[17] and for TopUML in [18]. As it will be shown in this paper, the authors resolve these issues by introducing the IDM approach and the supporting toolset.

This paper is considering the integrated domain modeling approach described in detail in [19]. This approach suggests starting the system analysis process from formally defined declarative and procedural knowledge with a perspective of integration with MDA. We are exploiting ontology and use cases for defining the knowledge model for a business domain. A knowledge engineer constructs the ontology and a business analyst constructs use cases. While doing so the use cases need to be validated in order to correspond to the ontology. This is an iterative process, because the ontology or the use cases have to be modified until they correspond to each other. The next step is acquiring the Business Model. When a Knowledge Model is constructed and verified, it is possible to generate the Business Model automatically using the TFM generation algorithm described in [19]. This algorithm utilizes the statistical parser to analyze the syntax of use case sentences and identify functional features for the TFM. Nevertheless, the TFM will have to be validated as well. If any changes are necessary, they will have to be done in the Knowledge Model and then the TFM can be regenerated. Additionally, within the Business and Requirements Model it is possible to derive the Business Processes and UML Use Case diagram from the TFM.

The IDM approach provides an elegant solution to the complexity of TFM construction. This approach proposes the following: 1) to step back to the knowledge level of the business system and to reuse the artifacts existing in a business environment as the input for CIM; 2) to acquire the initial CIM by means of automatic model transformations; 3) to allow the system analyst to iteratively validate, modify and improve the models by adding more details of the business processes to the initial CIM. For this to work in a real business environment for a business system modeling case there have to be tools to support the IDM approach. Moreover, these tools need to correspond to MDA standards and be based on available MDA frameworks. This will assure that the toolset is extendable and can be integrated with other modeling tools, thus becoming part of the MDA life cycle.

IV. Supporting Toolset

In this section, the supporting toolset for the IDM approach is introduced and the application of this toolset to acquire a CIM for a business system is discussed. As part of this research, the authors have implemented a prototype of the IDM toolset, which will also be discussed later in this paper.

In earlier studies [20] and [21], some suggestions have been made on what tool support would be necessary for the TFM approach. Moreover, considering the IDM approach (which substitutes the earlier TFM4MDA approach) first described in [19] and in more detail in the previous section, the scope of the required toolset can be discussed. The vision for an integrated MDA life cycle with TFM starting from construction of the CIM all the way to code generation would require a comprehensive toolset to support the process (Fig. 1 shows the vision).

The goal of the IDM Toolset is to acquire a formal and validated CIM in a form of a TFM based on formal knowledge about the business domain. As shown in Fig. 1, the toolset consists of 4 tools that can be used together to achieve this.

The users of this IDM Toolset are the knowledge engineer and the system analyst that can be several people or one person. The task of the user is to gather the business knowledge and record it in the form of Ontology and Use Cases. By Ontology one defines the declarative knowledge or, in other words. The dictionary of one's domain. This is where one can sort out the concepts, terms and also their meaning. This is done based on the existing business documentation and in close contact with the business team to clarify and validate the Ontology. In the IDM Toolset context, any tool that supports OWL standard for Ontology can be used, for example, Protégé. Later the Use Case Editor tool will use the acquired OWL artifact. Procedural knowledge is recorded in the form of Use Cases showing the scenarios with steps, alternative scenarios and conditions for some business domain objectives. Again working closely with the business team, the user needs to record the use cases using the Use Case Editor. When the Use Case artifact is acquired, it can be validated against the Ontology to check for unambiguity and consistency. For unambiguity the terms and concepts are compared with the nouns used in the use cases. By comparing Ontology properties and noun/verb combinations, consistency is checked. Next step in Use Case validation is to check if it corresponds to the tool meta-model. When the Use Cases are

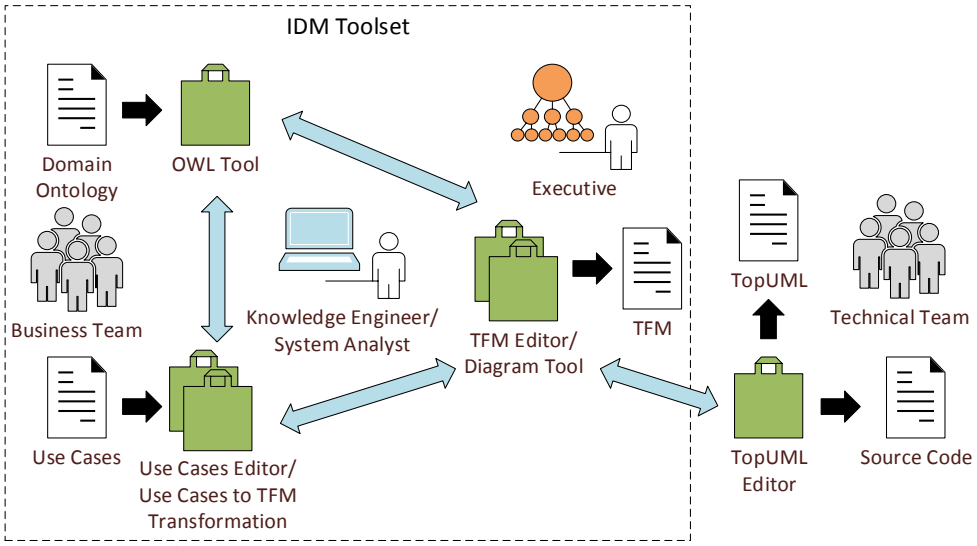

Fig. 1. TFM toolset components and artifacts. Here the planned toolset is shown with the involved people within a business environment to support the full TFM process within MDA life cycle. IDM toolset is a subset of this toolset (*dotted rectangle*), which includes OWL tool, Use Case Editor/Use Cases to TFM Transformation and TFM Editor/Diagram Tool.

validated, it is possible to automatically generate a TFM for the corresponding business domain. The acquired TFM artifact is available in the TFM Editor tool. The TFM artifact consists of the raw model and a diagram. The TFM Editor consists of 2 tools – the model editor and the diagram tool. Basically, these tools represent the same model in different ways – a model and a diagram; that is why in Fig. 1 these tools are merged.

During each stage of CIM construction cross-artifact validation should take place. For example, while creating a step in the Use Cases, if a user finds out that a term is missing or is incorrect in the Ontology he should do modifications. Another example is related to cause/effect relationships. These relationships are more apparent in the TFM; thus, after acquiring the TFM a user may find that some topological relationships are incorrect, so he should return to the Use Cases and do the corresponding modifications. This is an iterative process until Use Cases correspond to the Ontology and the TFM corresponds to the Use Cases. The TFM Editor also allows a user to identify the main functioning cycle, sub-cycles and add logical operations, which are not part of the model transformation. These elements add more details to the CIM and allow you to validate your models again from a different perspective – process topology and cycles. This is the CIM level within MDA, which represents the AS-IS state of a business system. At different stages of the CIM development, the business team should be consulted to validate the models and finally the CIM should be signed-off by the business executive. When the sign-off is done, the CIM is approved and the transformation to PIM/PSM can begin.

There are various MDA tools available in the area of PIM/PSM. It is possible to start with the UML diagrams (PIM level), then add some platform specific features with OCL (PSM level), and then based on this model it is possible to generate the source code. In the context of TFM at the PIM/PSM level, we use UML and more specifically TopUML, which is the UML profile. The process of transforming TFM to various TopUML models is described in related research [18]. Having a supporting tool for the UML profile is common practice, and MDA frameworks fully support this approach. Usually for a particular UML profile there is a corresponding tool for constructing models, which correspond to the UML profile. In case of TopUML, there has also to be such a tool, but on top of that, there will also be a set of model transformations from TFM to TopUML models. Furthermore, after TopUML is complete this tool will also have the feature to generate the corresponding source code. In Fig. 1, these features are the tasks of the TFM Editor tool with corresponding artifacts TopUML and source code.

The IDM Approach and Toolset can support various software development methods since use cases are widely accepted and used by the software engineering community as the starting point of software development. Moreover, it is possible to automatically acquire a graphical representation of the domain and the opportunity to perform further model transformations can be very useful. On the other hand, Ontology would be appreciated by more precise and sophisticated methods, when mistakes at the design phase are costly and developments should not start before the design is validated. Nevertheless, in the IDM approach Ontology is recommended, but not mandatory. This means that you can also only create use cases, generate the initial TFM and continue with CIM developments until you are satisfied with the result. This approach would be more suited for agile software development methods.

V. Architecture and Implementation

The authors stress the importance of using MDA standards for development of MDA approaches and the supporting toolsets. There are several attempts mentioned in the introduction by other researchers in the area of CIM construction and further model transformation with very poor integration into MDA standards. In the authors' opinion, this

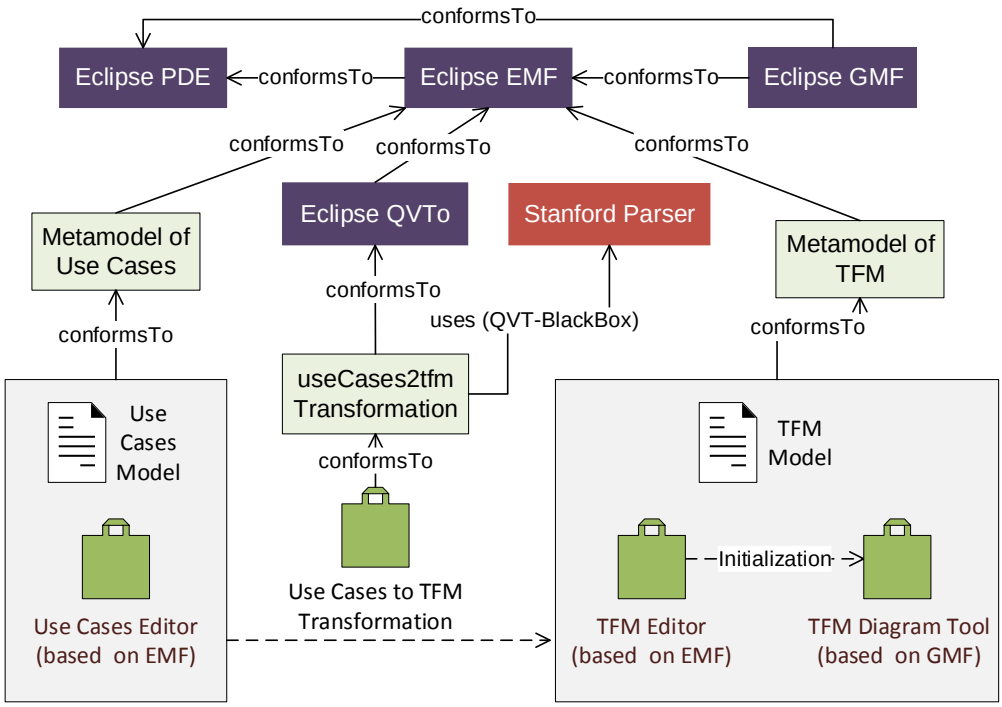

Fig. 2. Architecture of the IDM toolset. Dark rectangles on top represent the Eclipse framework used and also the Stanford Parser Java library. 3 lighter rectangles underneath show the development artifacts based on the frameworks and libraries created specifically for the IDM tools. The resulting toolset consists of 4 tools – Use Case Editor, TFM Editor, TFM Diagram Tool and Use Cases to TFM Transformation.

leads to a dead end for the approaches since they are cut off from the rest of the MDA developments. Even if they provide some integration possibilities, the lack of MDA standard based architecture leads to opacity. Because of this it is a key for the IDM approach and the toolset to be based on MDA standards and technologies.

Today MDA developments are based on Eclipse MDA frameworks, i.e., Eclipse Modeling Framework (EMF), Graphical Modeling Framework (GMF), Query View Transformation (QVT), etc. These frameworks are based on the MDA standards managed by OMG. In addition, Eclipse provides the Plug-in Development Environment (PDE) for developing Eclipse plug-ins; thus, it is possible to make use of all these frameworks and to develop your toolset in the same fashion based on Eclipse.

Fig. 2 shows the architecture of the developed components of the IDM toolset. This excludes the 3rd party OWL tool, for which we currently use the Protégé tool. The following Eclipse frameworks are used: PDE, EMF, GMF and QVT. The conformity is also shown. Since EMF and GFM allow generating Eclipse plug-ins, it conforms to the PDE, which can be later used to extend the functionality of the tools. Both GMF and QVTo (QVT Operational) conform to the EMF, which they are based on. In addition, a 3rd party statistical parser is used for natural language processing – Stanford Parser.

The following tools were developed by the authors: Use Case Editor, TFM Editor, TFM Diagram Tool and Use Cases to TFM Transformation. In contrast to Fig. 1, here the authors show the technical tools, which a user may not even be aware of while using the toolset. For example, the model transformation happens in the background after a left mouse click of the user and transformation execution. The Use Case Editor is initially generated by the EMF [22] from the metamodel of Use Cases, which is published in [23]. Later some modifications are done in the generated code, e.g., to change the default representation of the model in the model editor. These changes are marked with a special annotation so that possible changes in the metamodel could still be generated and the custom code would not be lost. The Use Case Editor tool is meant for constructing the Use Case model. Similarly, the TFM Editor tool is based on EMF and conforms to the latest TFM metamodel published for related TFM research in the doctoral thesis [24].

With this tool it is possible to edit the raw TFM model, but there is also a possibility to initialize a diagram to see the graphical representation of the TFM. This is done via the TFM Diagram Tool, which is based on the Eclipse GMF. By developing special configuration models, it is possible to generate the graphical modeling tool from the EMF based metamodel [25]. With both the TFM Editor and Diagram Tool you construct/edit the TFM model. As described previously in the IDM approach section, there is no need to construct the TFM from scratch and the initial model can be generated automatically from the Use Case model. This is handled by the Use Cases to TFM Transformation tool, which is based on the QVTo model transformation language. The transformation depends on natural language processing, so we use the Stanford Parser Java library [26]. This is integrated into the transformation tool via the QVT-BlackBox extension mechanisms suggested in the QVT standard and also Eclipse QVTo supports this approach [27]. The model transformation

and the black box are packaged into a single Eclipse plug-in and form the Use Cases to TFM Transformation tool. This also includes some Eclipse PDE developments to integrate with the Use Case Editor tool.

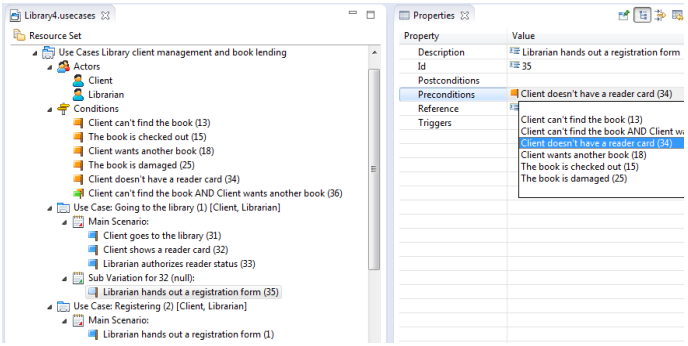

Fig. 3. Use Case Editor tool. This shows the use case model constructed by a user and properties view of a use case step, where it is possible to set preconditions.

Fig. 3 shows the Use Case Editor tool, which a user uses to define Actors, Conditions, Use Cases, Main Scenarios, Alternative Scenarios and their steps. More details on the development and application of this tool are published in [23] on the Use Case Editor particularly. The file extension of the Use Case artifact is ".usecases".

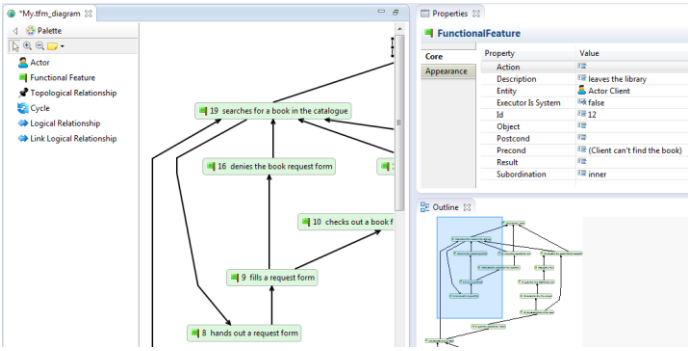

Fig. 4. TFM Diagram Tool. On the left-hand side there is a palette with the available objects for the diagram. In the middle there is a canvas with the diagram itself, and on the right-hand side there is a property pane and an outline view.

Fig. 4 shows the TFM Diagram Tool, where we can see part of the TFM generated from the corresponding Use Cases (from Fig. 3). A library business system is used for this example, but the full models are not described in this paper because of page limitations; and this paper does not focus on a case study, but rather on the approach and the toolset. As you can see in the screenshot, the palette of the tool allows you to create Actors, Functional Features, Topological Relationships, Cycles and Logical Relationships.

Properties of each element can be edited in the property view. When editing the diagram (file extension ".tmf_diagram"), also the underlying TFM model (file extension ".tfm") is changed by the diagram tool so that both artefacts are in sync.

Fig. 5 shows a code snippet from the QVTo model transformation for transforming Use Cases to TFM. The particular snippet deals with one of transformation's main tasks – creation of Functional Features in the TFM model. First, the transformation loops over all steps of the Use Case model. You can also see how we resolve the description and the entity for the Functional Feature from use case steps with methods – parseStepForDescription and parseStepForEntity, which in order call the Stanford Parser's library to find the noun and verb phrases. These methods are provided by the QVT-BlackBox library implemented in Java for this transformation (developed by the authors). Next, the transformation checks if the step is referenced, and if not then it prepares to create a Functional Feature. The model transformation needs to know the steps, which reference the current step so that it is possible to merge the pre-conditions and post-conditions (in case there are any). This is done by QVTo mapping and helper functions. This is a small part of the model transformation code, but in a similar fashion we deal with Actors, Functional Relationships, etc.

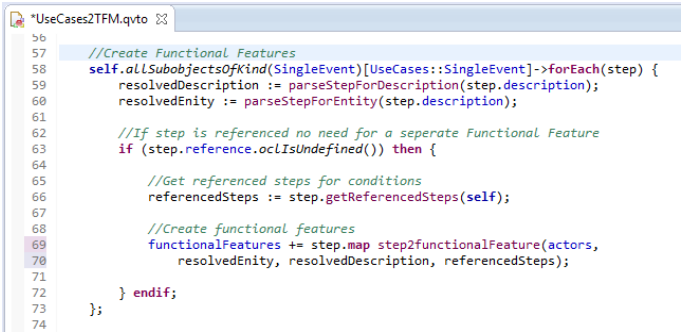

Fig. 5. Model transformation in QVTo. This is a small snippet from the entire model transformation code, which deals with generating functional features.

This model transformation from Use Cases to TFM enables the IDM toolset to fulfill its promise and automatically transform the domain knowledge expresses in use cases into a domain model in a form of TFM. Thus, a user acquires the graphical representation of a working business system by defining it with a common enterprise standard – use cases.

VI. Conclusion and Further Work

This research is part of Topological Functioning Model for Software Engineering (TFM4SE) research, which consists of the following: 1) Integrated Domain Modeling (IDM) approach tool, which allows defining the business processes with use cases, validating them against the ontology and then generating the domain model automatically in the form of a Topological Functioning Model (TFM); 2) TopUML (Topological UML) tool, which would enable a user to transform the TFM to TopUML, perform TopUML modeling and also generate the source code from the TopUML. The TopUML tool is future work in the context of TFM4SE research.

IDM tool, which is described in this paper, enables a system analyst to acquire and validate a domain model based on use cases and the ontology. This way it is possible to validate the business processes at the beginning of software development, check that they correspond to the ontology and also check the

functioning cycles of the processes. Based on the generated TFM, it is possible to create a TopUML to model a software solution, which corresponds to the domain model. By exploiting the domain model acquired by the IDM tool, the system analyst together with the business can validate the business processes before the actual software development starts.

REFERENCES

[1] R. Prieto-Díaz, "Domain analysis: an introduction," ACM SIGSOFT Software Engineering Notes 15.2, 1990, pp. 47–54.

[2] A. van Lamsweerde, "Requirements engineering: from craft to discipline," In *Proceedings of the 13th international Workshop on Early Aspects*, New York: Association for Computing Machinery, Inc., 2008, pp. 238–249.

[3] M. Kirikova, "Domain Modeling Approaches in IS Engineering," *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, 2011, pp. 388–406. http://dx.doi.org/10.4018/978-1-61692-874-2.ch018

[4] Eclipse [Online]. Available: http://www.eclipse.org/

[5] G. Fliedl, C. Kop, H. C. Mayr, A. Salbrechter, J. Vohringer, G. Weber, and C. Winkler, "Deriving static and dynamic concepts from software requirements using sophisticated tagging," *Data & Knowledge Engineering*, vol. 61, Iss. 3, 2007, pp. 433–448. http://dx.doi.org/10.1016/j.datak.2006.06.012

[6] J. Francu and P. Hnetynka, "Automated Generation of Implementation from Textual System Requirements," in *Proceedings of the 3rd IFIP TC 2 CEE-SET*, Brno, Czech Republic, Wroclawskiej, 2008, pp. 15–28.

[7] H. Kaindl, "Structural Requirements Language Definition, Defining the ReDSeeDS Languages," 2007. [Online]. Available: http://publik.tuwien.ac.at/files/pub-et_13406.pdf [Accessed: Oct. 7, 2013].

[8] K. Subramaniam, D. Liu, B. Far, and A. Eberlein, *UCDA: Use Case Driven Development Assistant Tool for Class Model Generation*, *Proceedin of the 16th SEKE*. Canada: Banff, 2004. [Online]. Available: http://enel.ucalgary.ca/People/eberlein/publications/SEKE-Kalaivani.pdf [Accessed: Sept. 18, 2010].

[9] J. Osis, "Topological Model Of System Functioning," (in Russian) in *Automatics and Computer Science, J. of Acad. of Sc.*, Zinatne, Riga, 1969, pp. 44–50.

[10] J. Osis, E. Asnina, and A. Grave, "Formal Computation Independent Model of the Problem Domain within the MDA," *Information Systems and Formal Models, Proceedings of the 10th International Conference ISIM'07*, Silesian University in Opava, Czech Republic, 2007, pp. 47–54.

[11] E. Asnina, "The Formal Approach to Problem Domain Modeling within Model Driven Architecture," in *9th International Conference on Information Systems Implementation and Modelling*, Czech Republic, Ostrava: Prerov, 2006, pp. 97–104.

[12] J. Osis and E. Asnina, "A Business Model to Make Software Development Less Intuitive," in *Proceedings of the 2008 International Conference on Innovation in Software Engineering*, Vienna, Austria. IEEE Computer Society CPS, Los Alamitos, USA, 2008, pp. 1240–1246.

[13] J. Osis, E. Asnina and A. Grave, "Formal Problem Domain Modeling within MDA," *Communications in Computer and Information Science (CCIS)*, vol. 22, Software and Data Technologies, Springer-Verlag Berlin Heidelberg, 2008, pp. 387–398.

[14] J. Osis, E. Asnina, "Derivation of Use Cases from the Topological Computation Independent Business Model," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions.* IGI Global, Hershey – New York, 2011, pp. 65–89. http://dx.doi.org/10.4018/978-1-61692-874-2.ch004

[15] J. Osis, E. Asnina, "Model-Driven Domain Analysis and Software Development: Architectures and Functions," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey – New York, 2011, 487 p. http://dx.doi.org/10.4018/978-1-61692-874-2.ch002

[16] J. Osis, E. Asnina, "Is Modeling a Treatment for the Weakness of Software Engineering?" in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey – New York, 2011, pp. 1–14. http://dx.doi.org/10.4018/978-1-61692-874-2.ch001

[17] J. Osis, E. Asnina, "Topological Modeling for Model-Driven Domain Analysis and Software Development: Functions and Architectures," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, IGI Global, Hershey – New York, 2011, pp. 15–39. http://dx.doi.org/10.4018/978-1-61692-874-2.ch002

[18] U. Donins, "Software Development with the Emphasis on Topology" in *Proceeding of 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009)*, vol. 5968 of LNCS, Springer, 2010, pp. 220–228.

[19] A. Slihte, J. Osis, U. Doniņš, "Knowledge Integration for Domain Modeling," in *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development*, China, Bejing, 8–11 June, SciTePress, Portugal, 2011, pp. 46–56.

[20] A. Slihte, "Implementing a Topological Functioning Model Tool," in *Scientific Journal of Riga Technical University*, 5. series., Computer Science, vol. 43, Riga, 2010, pp. 68–75.

[21] A. Slihte, "Transforming Textual Use Cases to a Computation Independent Model," in *MDA & MTDD 2010*, Greece, Athens, 2010, pp. 33–42.

[22] *EMF Developer Guide: The Eclipse Modeling Framework (EMF) Overview*, 2005. [Online]. Available: http://help.eclipse.org/ganymede/index.jsp?topic=/org.eclipse.emf.doc/references/overview/EMF.html [Accessed: Oct. 7, 2013].

[23] J. Osis, A. Slihte and A. Jansone, "Using Use Cases for Domain Modeling," in *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012)*, Poland, Wrocław, 2012, pp. 224–231.

[24] U. Doniņš, "Topological Unified Modeling Language: Development and Application," Doctoral thesis, Riga Technical University, 2012, pp. 224.

[25] F. Plante, Introducing the GMF Runtime, 2006. [Online]. Available: http://www.eclipse.org/articles/Article-Introducing-GMF/article.html [Accessed: Oct. 7, 2013].

[26] *The Stanford Parser: A statistical parser. The Stanford Natural Language Processing Group*, 2010. [Online]. Available: http://nlp.stanford.edu/ software/lex-parser.shtml [Accessed: Oct. 7, 2013].

[27] *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*, 2007. [Online]. Available: http://www.omg.org/cgi-bin/doc?ptc/07-07-07.pdf [Accessed: Oct. 7, 2013].

Armands Slihte currently is a Doctoral Student at Riga Technical University. In 2008, he graduated from Riga Technical University, Riga, Latvia, with a Bachelor's Degree in Computer Control and Computer Science. In 2010, he graduated from Riga Technical University with a Master's Degree in Computer Systems.

He currently works as a Researcher at the Faculty of Computer Science and Information Technology, Institute of Applied Computer Systems, Riga Technical University. The research focus of the authors is Model Driven Architecture (MDA), more specifically applying Topological Functioning Model (TFM) as the Computation Independent Model (CIM) within MDA.

E-mail: armands.slihte@rtu.lv

Juan Manuel Cueva Lovelle became a Mining Engineer from Oviedo Mining Engineers Technical School in 1983 (Oviedo University, Spain). He has a Doctoral Degree from Madrid Polytechnic University, Spain (1990). Since 1985 he has been a Professor in the Languages and Computers Systems at Oviedo University (Spain), and is an ACM and IEEE voting member. His research interests include object-oriented technology, language processors, human-computer interface, Web engineering, modeling software with BPM, DSL and MDA.

E-mail: cueva@uniovi.es