ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2009, Vol. 38, No. 2

TOWARDS IMPLEMENTING A FRAMEWORK FOR MODELING SOFTWARE REQUIREMENTS IN MAGICDRAW UML*

Darius Šilingas

Faculty of Informatics, Vytautas Magnus University
e-mail: Darius.Silingas@bpi.lt

Rimantas Butleris

Department of Information Systems, Kaunas University of Technology
Studentų St. 50, LT–51368 Kaunas, Lithuania
e-mail: Rimantas.Butleris@ktu.lt

Abstract. UML is considered to be *de facto* standard for software modeling. However, in software requirements analysis it is quite common to apply only use case and activity diagrams and focus on the textual requirements specification with some non-standard graphical illustrations. In this paper we present a framework for modeling software requirements consistently using multiple UML diagrams. We illustrate the application of this framework with the examples of different requirements artifacts based on a case study system MagicTest. We discuss how such a framework could be implemented in one of the most popular UML tools, MagicDraw UML, by using its powerful features for customizing the modeling environment, defining methodology wizards, specifying validation rules, analyzing model element relationships, and generating documentation based on user-defined templates. We recognize that our approach provides the foundation, which could and should be refined and extended for special cases of requirements analysis. Our work should be considered as a starting point for practitioners trying to adopt UML for requirements analysis and for scientists working on creating more detailed requirements analysis methods based on UML.

1. State of the Art in Requirements Analysis

Practitioners agree that requirements analysis is one of the most problematic and risky activities in software development. The output of requirements analysis is requirements specification, which should be reviewed and confirmed by all the stakeholders. Currently, most of the requirements specification approaches are based on textual documents including some cases for graphical illustrations. Since textual documents are difficult to understand and maintain, practitioners try to express the information in graphical illustrations. The basic problem with these graphical illustrations is that they are typically provided in different notations and have no concise interrelationships. This makes it problematic to turn the graphical models into the core of the requirements specification with textual descriptions being just additional explanatory elements, i.e. reversing the current state-of-the-art situation. Deploying successful requirements process in a concrete organization is an important issue [1, 2]. While companies continue to use text-based documents as major means for specifying and analyzing requirements, the graphical requirements modeling is getting increasingly more attention in industry. This trend increased after *Object Management Group* (OMG) standardized *Unified Modeling Language* (UML) [3], which has become *de facto* standard in industry. However, it is obvious that the potential of UML is not fully exploited in requirements analysis. The most popular requirements textbooks still introduce multiple requirement artifacts such as system context diagram and entity relationship diagram in outdated notations and do not emphasize how these artifacts could fit in UML-based modeling [4, 5]. This is understandable because UML has grown into a rather complex language – the recent version 2.2 defines 248 interrelated metaclasses (modeling concepts). Some papers present interesting discussions whether UML is becoming *universal* language instead of its primary purpose to be the unified language [6]. The complexity of UML allows many powerful applications, but also makes it difficult to learn the lan-

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "VeTIS" (Reg.No. B-07042)

guage and use it properly in practice. A typical modeler should use only a small subset of UML, which is relevant to his work. However, this subset is different depending on the application domain and modeler's role in software development process. It is also important to understand how to evolve UML models and relate the requirements artifacts properly. Since UML doesn't define modeling method, the practitioners lack guidance on how to apply it efficiently, and apply it only fragmentally loosing many benefits that UML provides. Practitioners and researchers propose different approaches for eliciting and analyzing software requirements. The most popular method used in modern requirements analysis is *use cases*. It was invented in *Ericsson*, popularized by Ivar Jacobson [7] and widely adopted in the industry. UML provides *Use Case* diagram for visualizing use case analysis artifacts. However, requirements analysis is not limited to use cases that capture only the end user level functional requirements. In order to specify precise requirements one needs to have a good understanding of the business domain. For this purpose, analysts create domain vocabularies, model business processes, business concept relationships, business rules and events, business object lifecycles. For modeling business processes, one can apply UML *Activity* diagram or *Business Process Modeling Notation* (BPMN), which is a new standard from OMG [8]. However, many practitioners still use the outdated IDEF notation [9] or non-standard modeling using vendor-specific symbols provided by tools like *Microsoft Visio*. For conceptual modeling analysts continue to apply the outdated *Entity Relationship* (ER) notation, which has been popular in database design since 70s [10]. This notation is pretty straightforward to map to UML *Class* diagram with a limited level of visible details. A lot of attention is paid to the business goals [11], business rules [12, 13], business object lifecycles, business roles and tasks in organization, which also can be done using simple extensions of UML [14]. The real-time and embedded system developers have defined a flavor of UML – *System Modeling Language* (SysML) [15], which defines *Requirements* diagram, enables capturing various non-functional and detailed functional requirements and defining specific links between requirements and other model elements through some simple extensions of UML. These ideas are also valid for enabling requirements traceability in typical software development projects. Some of the other requirements artifacts that might be expressed in UML are system context diagram, data flows, user interface navigation schemas and prototypes [16]. Some articles have already discussed the suitability of UML for modeling requirements [17, 18]. The possibility of having all this information in a single concise UML model inside a state-of-the-art modeling tool reveals a huge potential for more concise requirements analysis and management, which could be achieved by validating model for correctness and completeness, tracing requirements artifacts to design and implementation elements, analyzing model metrics, and generating documentation reports. Although UML provides means for expressing different requirements artifacts, practitioners also need some methodology guidance how to start and evolve requirement models. In academic community, researchers propose various detailed and focused requirements development methods [19-21]. However, most of the methods resulting from academic research are too complex for practical applications and solve just specific issues. A simple, adaptable, and easy-to-implement framework for requirements modeling with samples on a realistic case study gives much more value for practitioners. In [22], one of the authors of this paper discussed the framework for creating UML models for *Model-Driven Development* (MDD). In [23], this framework was revised by shifting focus on the details of the specific parts of this framework related to requirements analysis. It also presented the mapping of the most common requirements artifacts to UML. In this paper we would like to refine the framework for modeling software requirements and discuss how it could be implemented in one of the most popular modeling tools – *MagicDraw UML*. We believe that the aspect of supporting this framework in an industrial tool is a very important issue, which needs thorough research and presentation in order to make this framework valuable for both practitioners willing to use it in practical projects and researchers willing to refine it by adding more details or modifying it for a domain-specific usage.

2. A 10-Step Framework for Modeling Software Requirements

While in [23] we have discussed which of the MDD framework elements are relevant artifacts resulting from the requirements analysis, here we would like to refine this framework by leaving only the requirements analysis tasks, decomposing some of them into more fine-grained tasks, and emphasizing which of these tasks are performed in an iterative manner. This refined framework is represented in Figure 1. It includes 10 tasks, each of which produces different type of requirements artifact in UML model. For simplicity reasons, the activity diagram defines these tasks as sequential, but in practice it is quite common to do some of them in parallel. The first four steps should be considered as domain modeling activities that define the context for building software systems, the next four steps should be considered as requirements modeling activities, and the last two are design activities in theory, but in practice it is pretty common to assign them to requirements analysts.

We recommend starting domain analysis by identifying the domain concepts and relationships between them. For this activity, we propose to use a simplified UML class diagram, which should be limited to classes hiding their attribute and operation compartments and using only simple associations with names and role multiplicities. An example of such an artifact is

given in Figure 2. A concepts map is sometimes called a visual vocabulary of business concepts. Modelers should add some textual description about each of the concepts in the model. A modeling tool should have functionality for running a report producing a document of a desired format (XML, HTML, RTF) including the diagrams and model element descriptions in a desired style, which is typically passed to the tool as a user-defined template.

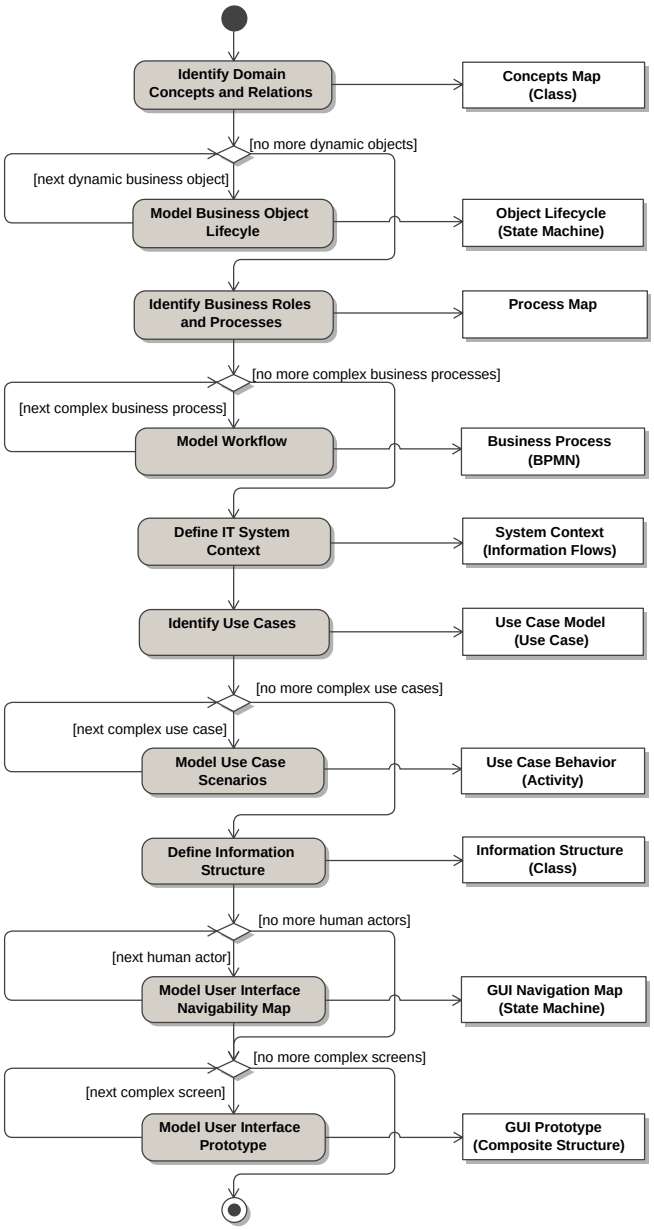

Figure 1. UML activity diagram visualizing the revised framework for modeling software requirements defined as a sequence of 10 tasks, each of which produces a specific type of requirements artifact in UML model

Some of the business concepts (modeled by classes) represent dynamic business objects that have complex lifecycles. For each of such concepts, one should define a *State Machine*, which should be assigned as a *Behavior* for the *Class* expressing the business concept. A modeler should create a separate state machine diagram for every dynamic business concept. In Figure 2, the dynamic business concepts are shown in black color. In Figure 3, a lifecycle of *Test* concept is represented in a state machine diagram. It is important to emphasize that the state names are also a part of terminology and should be used consistently in the other model elements, e.g. names of tasks in business processes. A modeler should define triggers on all the transitions between the states. In business modeling, it is common to use *Signal* that in most cases corresponds to an action of some business role. Also, *Timer* and *Property Change* triggers are used to express states changes according to time- or data-based business rules. It is also possible to define inner triggers that happen inside one state and do not initiate transitions to the other states.

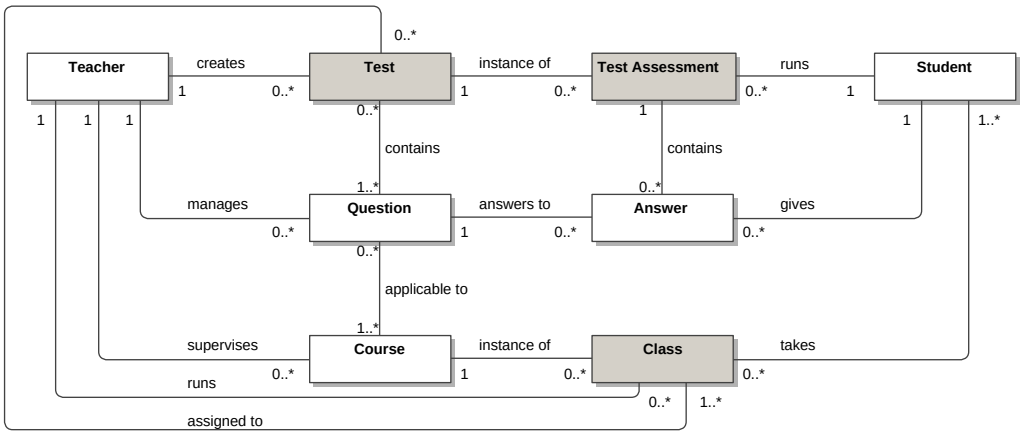

Figure 2. Concepts Map: UML class diagram visualizing business concepts and relationships

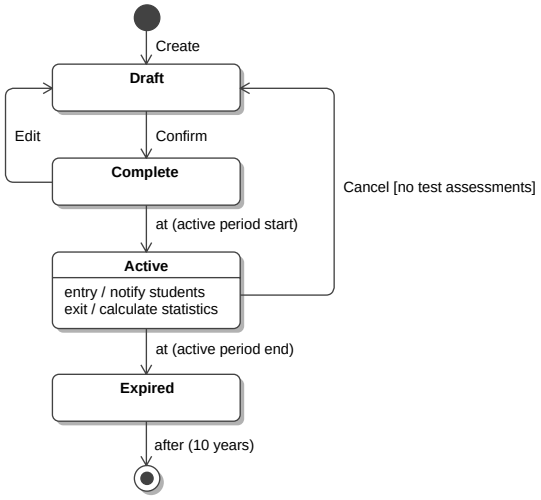

Figure 3. UML state machine diagram visualizing *Test* business object lifecycle

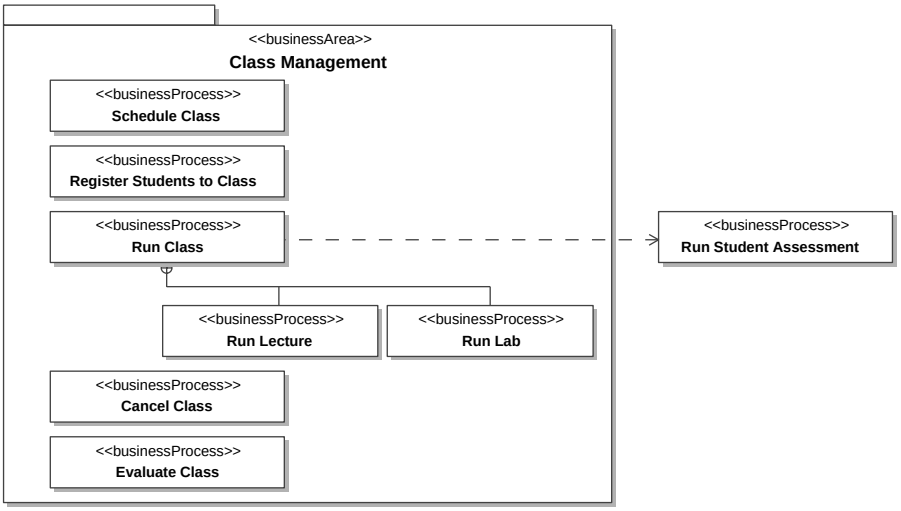

Figure 4. Process map: business areas, processes, and their relationships

Business analysts consider business processes as their major analysis artifacts. The goal of the BPMN initiative was to standardize modeling workflows, but the authors of BPMN decided to not cover the aspects of business areas, business roles, and the structural relations of the business processes. For covering these aspects, we suggest to use a couple of specialized diagrams: an extension of use case diagram for modeling business roles and their participation in business process, and an extension of class diagram for modeling business areas and business process structure, see Figure 4.

Once business roles and processes are identified, a modeler can specify the workflow of the selected complex business processes. We recommend using BPMN for this purpose. At the moment of writing this paper, BPMN does not have a formal metamodel and is typically implemented as a UML profile providing extensions to the activity diagram. This also makes it possible to relate business process workflow elements to the non-BPMN artifacts expressed in UML such as those created in the previous steps. The business processes are usually modeled in two forms: "*as is*", representing the current situation, and "*to be*", representing the target situation that should be reached after automating or refactoring some activities in the process. In the context of software development, it is typical to denote the places in the business process that are planned for automation by IT systems. An example of such business process workflow is given in Figure 5.

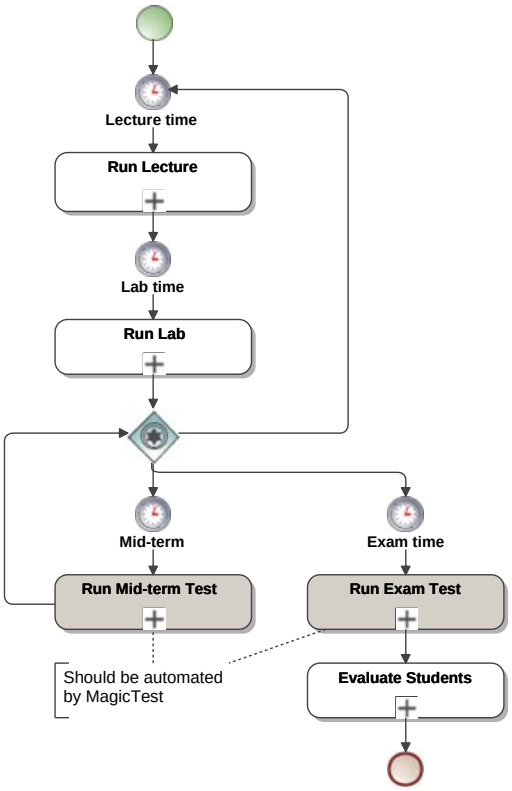

Figure 5. The workflow of the process *Run Class* in BPMN

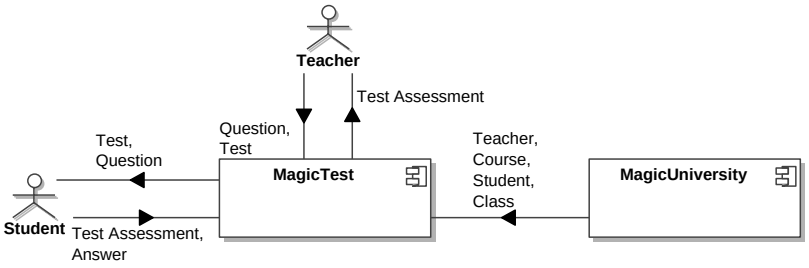

Figure 6. UML information flows diagram representing system context

In many data-centric applications, it is very important to model information flow diagrams showing data flows between different classifiers. A particular kind of such a diagram, which indicates the information flows between the target system and the outside entities (human actors and integrated systems), is called a *system context diagram*. It might be considered as the first artifact that an analyst can do for defining the scope of the system from the data point of view. An example of a system context diagram is given in Figure 6.

Once we have identified the places in the business process(es) that should be automated (and possibly a system context diagram as well), we can identify the actors (categories of system users) and define the use cases – the pieces of the system functionality that brings value for the actors. If the system contains a large number of use cases, it is common to group them into packages and analyze the details of each package in a separate diagram. An example of a use case diagram for a separate package is given in Figure 7. It is important to understand that the use case diagram

captures only the functionality that the end-user needs from the system. The non-functional requirements or detailed functional requirements are not captured in the standard UML diagrams. The simplest way to capture those requirements is to describe them in text documents and include references to the use cases, their scenarios, etc. Another approach is to create specific UML extensions for requirements modeling, i.e. introduce stereotypes for each important requirements type with tag definitions for the custom properties and define the types of links for tracing the requirements, e.g. *derive*, *satisfy*, *support*.

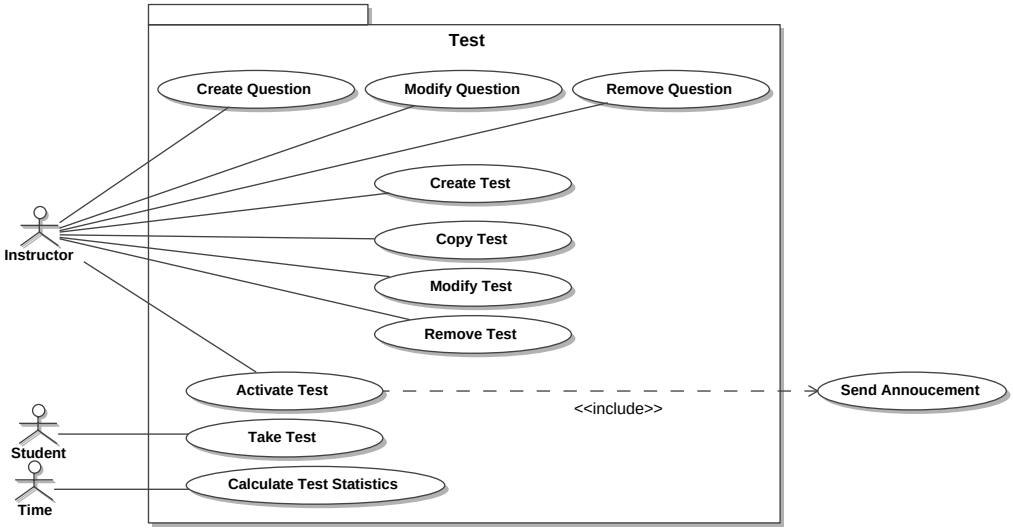

Figure 7. UML use case diagram focusing on the use cases of the *Test* package

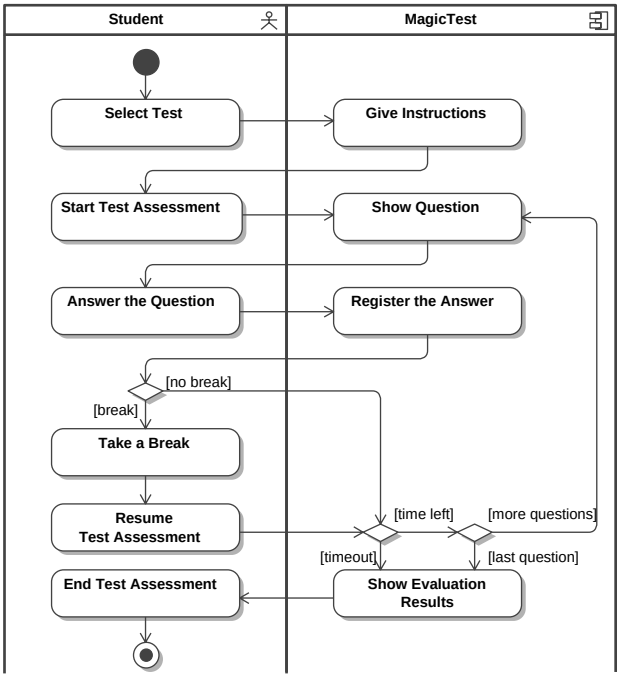

Figure 8. UML activity diagram representing behavior scenarios for the use case *Run Test Assessment*

Having an overall view of the use cases, an analyst can identify the use cases that contain important business or operational rules causing workflow branching. For such complex use cases, the analysts should model their scenarios applying UML activity diagram. However, it is important to avoid a common mistake of trying to make an activity diagram for every use case – they should be done only for the complex use cases. An example of such a use case workflow diagram is given in Figure 8.

Another aspect on which system analysts work in some projects is a definition of the data structure. It can be done using conventional UML class diagrams. If necessary, UML object diagrams can also be used for defining samples for explanation or testing of the data structure defined in UML class diagrams. Since the focus here is on the data structure, class operations compartment can be hidden in the diagram. Comparing to the conceptual analysis, more elements are used here: attributes and association end specifica-

tions, enumerations, and generalization. Although such a model is considered to be a part of the design, in practice it is quite often created and maintained by a requirements analyst. A sample a data structure diagram is shown in Figure 9.

The last two requirements artifacts for which system analyst might be responsible are user interface navigation schemas and prototypes. The prototype itself can be mapped to UML composite structure diagram. However, when focusing on separate screen prototypes, developers sometimes loose the big picture – which screens can be used by an actor and what are the possibilities to navigate from each screen to the others. For capturing this information, a modeler can create GUI navigation map (a separate one for each actor) using UML state diagram, where each state represents a screen, in which an actor is at the moment, and the transition triggers represent GUI events such as mouse double-click or pressing some button. Again, this is considered to be a part of design, but in practice it quite often falls on the shoulders of a requirements analyst. An example of a GUI navigation schema is given in Figure 10, and an example of GUI prototype is given in Figure 11.

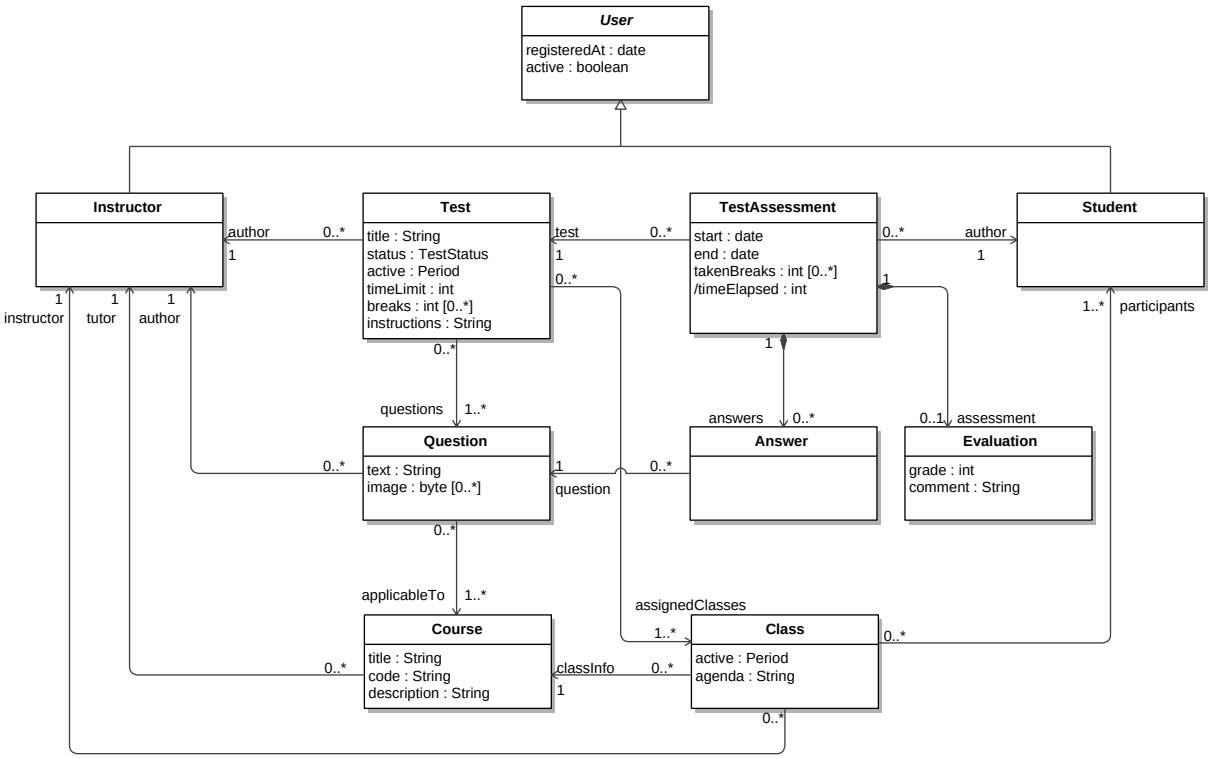

Figure 9. UML class diagram representing data structure

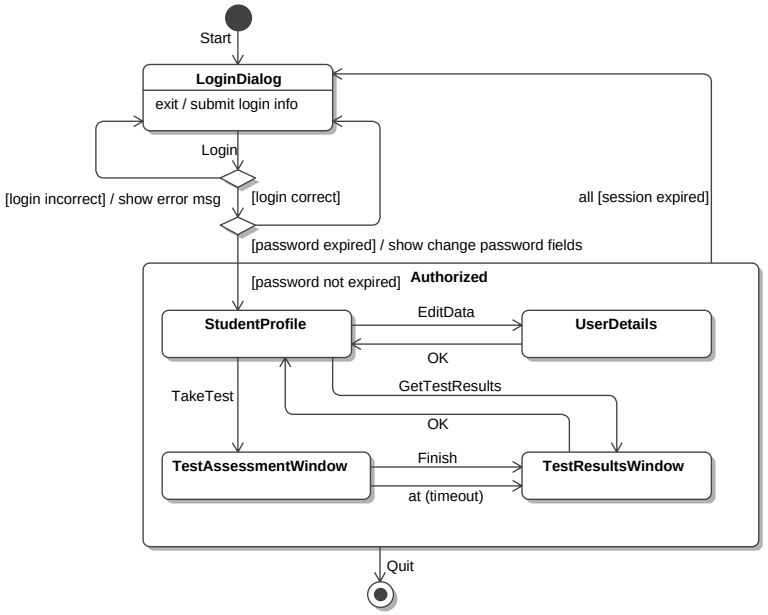

Figure 10. UML state machine diagram showing a user interface navigation map for *Student* actor

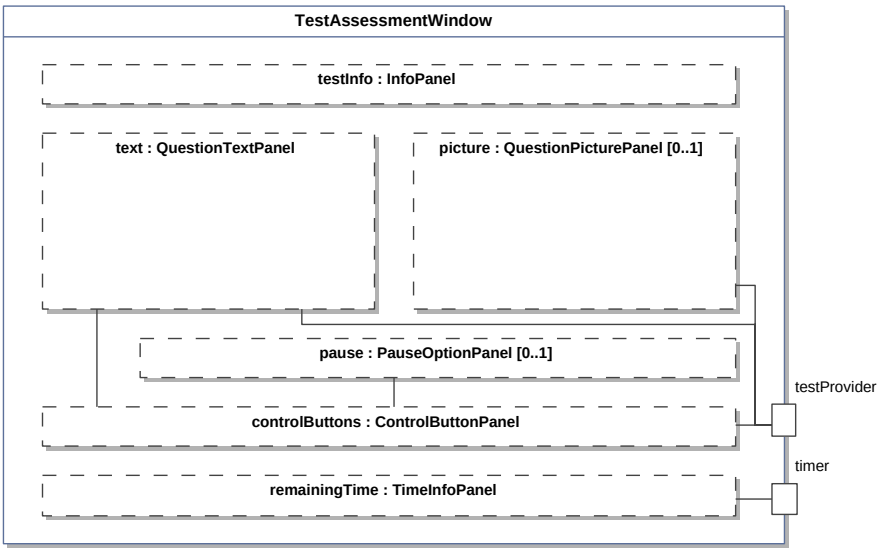

Figure 11. UML composite structure diagram showing a prototype of the *Test Assessment Window* GUI dialog

Finally, we want to emphasize that the requirements analysis work should be iterative and incremental. Also, the ordering of the modeling tasks might be different based on the taken approach, or some steps might be omitted if not relevant to a particular software development project.

3. The Principles of Implementing the Framework in MagicDraw UML

In order to support the framework for modeling software requirements in a particular UML modeling tool, we face the following problems:

1. UML modeling environment is too complicated for the software requirements analyst, who is willing to use only a subset of UML. There should be a way to hide the not used UML modeling functionality. Also there should be a way to introduce simple extensions to UML that are necessary for requirements analysis.
2. An analyst needs some guidance how to start and continue modeling according to the framework.
3. It would be very helpful to have some possibilities to check if the requirements model or a particular part of it is consistent, i.e. doesn't break some modeling rules or conventions, and if the model or a particular part of it is complete, i.e. it contains all the necessary information.
4. It is necessary to have easy tools for creating, maintaining, and analyzing the requirements artifacts traceability information.
5. It is necessary to be able to add additional textual documentation and output the graphical and textual information in a document, which format is widely acceptable. This is necessary since the reviewers of the requirements specification will not have the modeling tool and the possibility to explore the model in its native UML format.

Although the number of UML tools available in the market is very large, the mentioned problems can be solved only in a few of them. We believe that the most elegant solutions for implementing the framework are enabled in MagicDraw UML, which is widely regarded as the most UML-compliant tool and has very powerful features for configuring and extending its modeling environment. In the next subsections we will discuss the principles of how to implement the requirements framework in MagicDraw and provide the solutions for the identified problems.

3.1. Customizing UML Modeling Environment for Requirements Analyst

MagicDraw UML provides the following capabilities for customizing the modeling environment according to the needs of the modeler:

- *User perspectives* – setups of what functionality is visible and what is hidden in menus, contextual menus, diagram toolbars, smart manipulators, etc. A custom user perspective for requirements analyst would be a perfect way to limit the environment to only those features that he needs to use in his work.
- *Custom diagrams* – a modeler may define his own diagram, which is using a subset of elements from a standard UML diagram or a set of specific extensions of UML. For example, in order to support the requirements framework, one may define two different versions of simplified class diagram – *Concepts Map* and *Data Structure*, and a specialized diagram *Process Map*.
- *Environment and project options* – tool configuration according to the user preferences (e.g. use diagram grid or not) or project conventions (e.g. what fill color should be default one for a class).
- *Template projects* – a sample project used as a starting point for other projects in order to maintain the same model structure, reference a consistent set of libraries, and reuse project options.

- *Element customization* – a feature, which enables turning UML stereotype into a virtual first-class modeling concept and hiding the UML concept behind it. Although we discourage using this feature too often (it brings modelers back to speaking different languages), it is very useful in the cases when for domain modelers UML is too technical. We especially recommend using it for creating the specialized elements and diagrams for the business process modeling.

3.2. Creating Modeling Guidance Using Methodology Wizards

For particular steps of the framework for requirements modeling the modeler should apply a defined sequence of discrete actions. In such cases a modeler may specify and enable the methodology wizards taking modeler through a number of steps for creating the model. This is one of the MagicDraw features allowing model-based extension of the modeling environment.

We propose to do the use case analysis in the following steps:

1. Specify the name of the system.
2. Identify the actors.
3. Identify the use cases.
4. Relate each actor to its use cases.
5. Structure use cases into packages.
6. Detail the flow of complex use cases with the activity diagrams.
7. Create the views (diagrams) for visualizing the most important fragments (e.g. the use cases and their relationships for a specific package) of the use case models.

For automating this sequence of actions we could define a specialized activity diagram, which MagicDraw would turn into a wizard, which takes modeler step-by-step for accomplishing the use cases modeling activity. An example of such a specialized wizard is given in Figure 12, and an example of the resulting methodology wizard is given in Figure 13. For facilitating the usage of the framework for software requirements modeling, we suggest to define multiple wizards for guiding the modeler through the particular activities.

3.3. Validating the Requirements Model

MagicDraw UML defines the mechanisms for creating validation rules and validating the user model based on these rules. The simple extensions of MagicDraw validation mechanisms are enabled through three stereotypes: *validationRule*, *validationSuite*, and *automatedValidationSuite*. A validation rule is specified by a stereotype *validationRule*, which is based on the *Contstraint* metaclass. For specifying a validation rule, the modeler needs to enter the properties (*name*, *constrained element*, and *specification* are standard *Constraint* properties, while *severity*, *error message*, *abbreviation*, and *implementation* are tag definitions from the *validationRule* stereotype) that are described in Table 1.

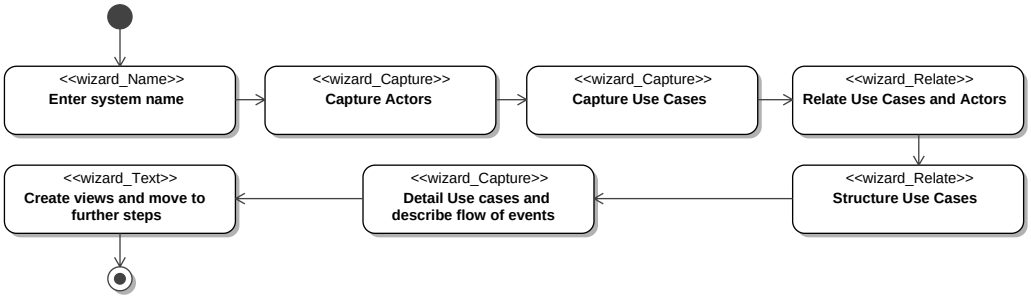

Figure 12. A specialized activity diagram that is used for creating a methodology wizard

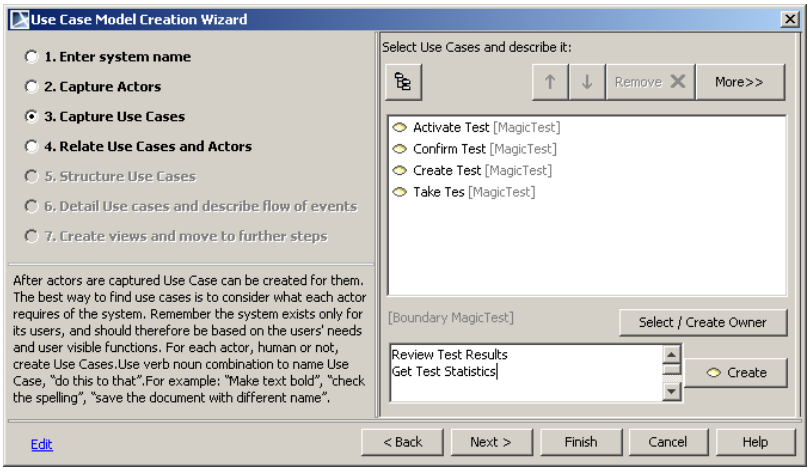

Figure 13. Use Case Model Creation Wizard that was created automatically based on the activity diagram in Figure 12

Table 1. Description of validation rule properties

Property	Description
Name	*A title for representing the rule in the model repository and diagrams*
Constrained element	*UML metaclass or a stereotype to which the validation rule applies*
Specification	*OCL-based specification of invariant rule that should be true for valid elements*
Severity	*A level of validation error importance: Debug, Info, Warning, Error, Fatal*
Error message	*The message text that should be displayed for the modeler for explanation why a model element is invalid*
Abbreviation	*A short name*
Implementation	*Reference to Java class implementing a specific interface for model validation and solving the validation problems*

Validation rules are grouped into validation suites – packages stereotyped by either *validationSuite* or *activeValidationSuite*. The former is used when there is a need to validate the model only occasionally based on modeler's wish, while the later is used when there is a need to validate the model actively in the modeling progress. An example of a validation rule specification in OCL 2.0 is presented in Figure 14, and a validation error in modeling environment resulting from this rule is demonstrated in Figure 15.

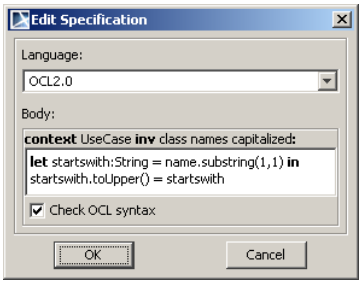

Figure 14. OCL 2.0 specification of the validation rule checking if a use case name starts with a capital letter

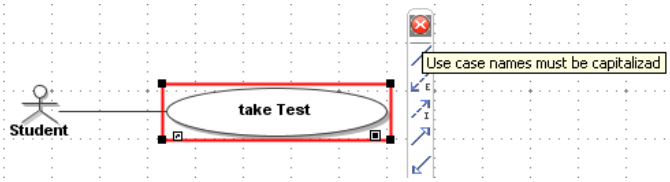

Figure 15. A screenshot indicating a validation error resulting from the validation rule specified in Figure 14

For supporting the framework for modeling software requirements it would be very useful to define a number of reusable validation rules and use them in various simple or active validation suites. Although in UML a package owns its inner elements, the validation suite package may use outgoing *element import* relationships to include the validation rules from different places in the model.

3.4. Building Model Element Relationships Matrices

UML defines a number of element relationships – *dependency*, *abstraction*, *realization*, *usage*, *association*, *generalization* and others. A modeler can use them for relating the elements of his model, but it is not easy to trace the elements, analyze the traceability information, and get an overall view about particular relationships. For this purpose in MagicDraw UML, one can use either analysis tools for tracing the related elements or set up and generate a number of dependency matrices. The setup is pretty easy – a modeler just needs to say from which package(s) in the model what kind of concepts should be taken and what kind of relationships should be traced. A fragment of MagicDraw UML screenshot showing both the setup and the generated matrix is given in Figure 16. We suggest building a number of such diagrams for tracing different aspects, e.g. data usage in business processes, information usage in user interface elements, use case realizations in components.

3.5. Generating Requirements Specification Documents

MagicDraw UML supports creating document templates in their target formats – simple text, rich text (RTF), Open Office documents with Velocity-based scripts for getting data from the model. A fragment of such a template is given in Table 2 and the output from a documented model based on this template is given in Table 3.

Obviously, for supporting the framework for software requirements modeling, multiple documentation templates must be developed – business concepts vocabulary, use case specification, business processes, data structure, user interface model, etc.

Table 2. A document template for retrieving the business vocabulary – business concepts and their textual descriptions

Concept	Description
#forrow ($class in $sorter.sort($Class)) $class.name	*$report.getComment($class) #endrow*

Table 3. The output from MagicDraw documentation engine generated based on the template given in Table 2 for the business concepts represented in Figure 2 and documented with textual descriptions in the model

Concept	Description
Answer	*Student's input answering to a particular question included in the test.*
Class	*A particular running of a discipline, which is taught by a teacher to a number of students who have registered for it.*
Course	*A discipline, which could be taught multiple times.*
Question	*A statement asking for the answer in order to assess student's knowledge or skills based on the course contents. Each question should be applicable to at least one course.*
Student	*A member of university who aims to get a qualification degree by participating in a number of classes.*
Teacher	*An employee of the university who is responsible for supervising courses and teaching classes. A teacher can create questions and compose tests that are assigned for one or more of his classes.*
Test	*A setup of the test that includes the period in which the test is active. i.e. available for assessments, the classes for which it is assigned, a selection of questions (all of them should be applicable to the course that is instantiated by the assigned classes), instructions and some other properties.*
Test Assessment	*A particular trial of the student to take the test including the answers the questions defined in the test, start and end time, and the evaluation.*

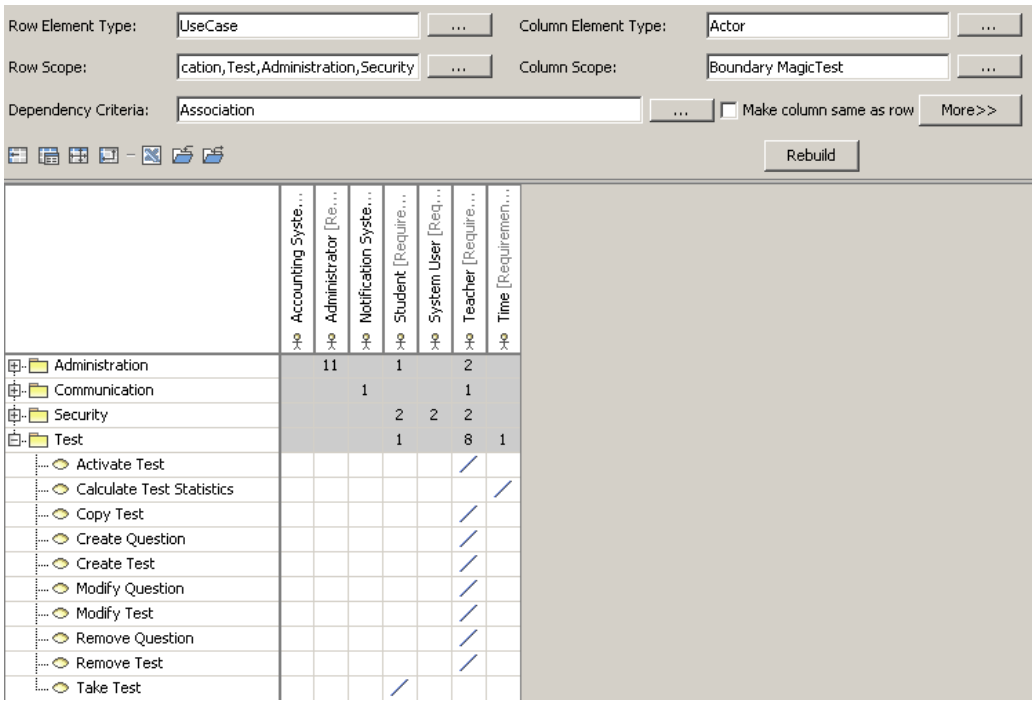

Figure 16. A set-up and output of Dependency Matrix visualizing associations between actors and use cases

4. Conclusions

We have presented a short review of modern requirement analysis issues emphasizing motivation for more consistent application of UML for requirements modeling. We have introduced a framework for software requirements modeling and demonstrated its steps with consistent requirements modeling artifacts for a case study system MagicTest. We have also discussed the basic principles of how this framework can be implemented in MagicDraw UML. The presented framework gives the generic guidelines for the software analysts. For a specific requirements modeling in a particular project or organization, it is necessary to add more details and implement multiple artifacts based on the presented principles. The research community should also consider this work as a starting point for creating more detailed and more formalized methods for requirements analysis.

In the future, we plan to work on more detailed guidance for requirements modeling framework, and implementation of this framework in MagicDraw UML – development of validation rules, documentation templates, and customization of the environment. We also plan to implement a prototype of MagicTest system for demonstrating the power of UML and model-based development approach, part of which is model-based requirements analysis.

References

[1] **J. Aranda, S. Easterbrook, G. Wilson.** Requirements in the wild: How small companies do it. 15*th IEEE International Requirements Engineering Conference* (*RE* 2007), 2007, 39-48.

[2] **M.C. Panis, B. Pokrzywa.** Deploying a System-wide Requirements Process within a Commercial Engineering Organization. 15*th IEEE International Requirements Engineering Conference* (*RE* 2007), 2007, 295-300.

[3] **Object Management Group**. Unified Modeling Language: Superstructure. *Formal Specification, version* 2.2, 2009.

[4] **K.E. Wiegers.** Software Requirements. 2*nd edition*, *Microsoft Press*, 2005.

[5] **E. Gottesdiener**. The Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements. *GOAL/QPC*, 2005.

[6] **G. Engels, R. Heckel, S. Sauer**. UML – A Universal Modeling Language? *In M. Nielsen, D. Simpson (Eds.): Application and Theory of Petri Nets* 2000, 21*st International Conference,* (*ICATPN* 2000), *Aarhus, Denmark, June* 26-30, 2000, *Proceedings, Lecture Notes in Computer Science*, 2000, *Vol*.1825, 24-38.

[7] **I. Jacobson**. Object-Oriented Software Engineering. *Addison Wesley Professional*, 1992.

[8] **Object Management Group**. Business Process Modeling Notation Specification. *Final Adopted Specification, version* 1.0, 2006.

[9] **O. Noran**. UML vs. IDEF: An Ontology-oriented Comparative Study in View of Business Modelling. *Proceedings of the* 6th *International Conference on Enterprise Information Systems* (*ICEIS* 2004), *Porto*, 2004, *Vol*.3, 674-682.

[10] **P.P.-S. Chen**. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems* (*TODS*), *Vol*.1 (1), 1976, 9-36.

[11] **A. van Lamsweerde**. Goal-Oriented Requirements Engineering: A Guided Tour. 5*th IEEE International Symposium on Requirements Engineering*, *Toronto*, *August*, 2001, 249-263.

[12] **K. Kapocius, R. Butleris.** Repository for Business Rules Based IS requirements. *Informatica, Vol*.17, *No*.4, 2006, 503-518.

[13] **E. Pakalnickiene, L. Nemuraite.** Checking of conceptual models with integrity constraints. *Information technology and control*, *Vol*.36, *No*.3, 2007, 285-294.

[14] **M. Penker, H.-E. Eriksson.** Business Modeling With UML: Business Patterns at Work. *Wiley*, 2000.

[15] **Object Management Group**. Systems Modeling Language. *Formal Specification, version* 1.0, 2007.

[16] **T. Danikauskas, R. Butleris, S. Drąsutis.** Graphical user interface development on the basis of data flows specification. *Computer and Information Sciences – ISCIS* 2005, 20*th International Symposium, Istambul, Turkey, October* 26-28, 2005, *Lecture Notes in Computer Sciences*, *Vol*. 3733, 2005, 904-914.

[17] **M. Glinz.** Problems and Deficiencies of UML as a Requirements Specification Language. *Proceedings of the* 10*th International Workshop on Software Specification and Design*, 2000, 11 – 22.

[18] **S. Konrad, H. Goldsby, K. Lopez, B.H.C. Cheng**. Visualizing Requirements in UML Models. *Proceedings of the International Workshop on Requirements Engineering Visualization* (*REV* 2006), 2006.

[19] **R. Butkiene.** Method for Specification of Functional Requirements for Information System. *Doctoral Dissertation, Kaunas University of Technology*, 2002.

[20] **J. Castro, M. Kolp, J. Mylopoulos.** Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems, Elsevier*, *Vol*.27, *Issue* 6, 2002, 365-389.

[21] **H. Behrens.** Requirements Analysis and Prototyping using Scenarios and Statecharts. *Proceedings of ICSE* 2002 *Workshop: Scenarios and State Machines: Models, Algorithms, and Tools*, 2002.

[22] **D. Silingas, R. Vitiutinas.** Towards UML-Intensive Framework for Model-Driven Development. *B. Meyer, J.R. Nawrocki, and B. Walter (Eds.): Second IFIP TC* 2 *Central and East European Conference on Software Engineering Techniques* (*CEE-SET* 2007), *Poznan, Poland, Lecture Notes in Computer Science, Vol*. 5082, 2008, 116-128.

[23] **D. Silingas, R. Butleris.** UML-Intensive Framework for Modeling Software Requirements. *Proceedings of International Conference on Information Technologies* (*IT* 2008), *Kaunas*, 2008, 334-342.

Received February 2009.