ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2011, Vol.40, No.3

MODEL-DRIVEN PLUG-IN DEVELOPMENT FOR UML BASED MODELING SYSTEMS

Ruslanas Vitiutinas, Darius Silingas, Laimutis Telksnys

Faculty of Informatics, Vytautas Magnus University,
Vileikos str. 8-409, Kaunas, Lithuania
e-mail:darius.silingas@nomagic.com, ruslanas.vitiutinas@nomagic.com, telksnys@ktl.mii.lt

crossref http://dx.doi.org/10.5755/j01.itc.40.3.627

Abstract. UML is the main modeling language used in model-driven development (MDD). In many cases, UML-based modeling systems need to be extended by plug-ins to support different modeling approaches. This paper proposes a conceptual framework for model-driven development of plug-ins, which enables reuse of UML modeling capabilities for defining executable plug-in models. This approach suggests that UML-based modeling system should provide for their users a set of extension profiles that make up Application Modeling Interface (AMI), which is MDD alternative to Application Programming Interface (API). The paper describes three cases of AMI – model validation, methodology wizards, and model patterns – and sample plug-in models based on them. The presented samples are implemented in MagicDraw, which is one of the most popular UML-based modeling systems worldwide. The paper also discusses the benefits and drawbacks of the proposed approach and its further research directions.

Keywords: Model-driven plug-in development, UML, MagicDraw, Application Modeling Interface, model validation, model patterns.

1. Introduction

The plug and play architecture is a well-known approach to building integrated devices that are extendible with plug-in components supporting defined interfaces. This approach has been successfully adopted in software industry. Many software products provide open API (Application Programming Interface), which allows its users to implement custom plug-ins adding missing capabilities to the product functionality. In some cases, the open API offers only small set of possibilities, and in other cases the products have a core platform with rich open API for adding plug-ins. The latter approach has been successfully implemented in Eclipse platform [6] and was one of the main reasons why it quickly became the most popular toolkit for software developers [18]. For those systems that have a very large and non-homogenous user base, it is extremely important to provide extension mechanisms so that the users can implement capabilities for their specific needs themselves. Such extension mechanisms are very important in modeling systems, because modeling approaches vary a lot between organizations and there are little standardization in modeling methods – instead standardization bodies focus on languages like Unified Modeling Language (UML), Business Process Model and Notation (BPMN), System Modeling Language (SysML) and others that are independent of particular modeling approach or application domain. As a consequence, modelers applying these languages typically have to tailor the modeling systems to support their modeling approach and extend the languages for their domain needs. Taking into consideration a huge and non-homogenous modelers base, the ease of developing these extensions are critical not only to success of modeling system products but also to the success of MDD paradigm itself – any good initiative can fail if there are no proper tools in practice.

In 2003, Object Management Group (OMG) introduced the Model Driven Architecture (MDA) initiative, which promoted using modeling as the main means for producing executable software [33]. At that time, Unified Modeling Language (UML) was already considered to be *de facto* standard in software modeling and it has maintained this position until now. Many commercial and open source modeling systems using UML as the built-in metamodel have been developed. The most popular modeling systems such as MagicDraw, IBM Rational Software Architect, Enterprise Architect and others provide open Application Programming Interface (API) enabling users to create plug-ins for addressing their custom needs. A large number of such plug-ins have been created for practical or research purposes. However, API defines programming, not modeling interface. From the perspective of MDA, vendors of the modeling systems should practice the well-known "*eat your own dog food*"

principle [21] and promote modeling instead of programming as the means for extending the capabilities of the modeling systems itself. Such an approach to extending modeling systems is very natural from modelers' viewpoint – they know modeling language that is supported by the modeling system and have modeling skills but they might not know particular programming language like Java, C++ or other in which open API is provided. In case of extending modeling system via open API, it is typical to assign this job to a special team, which has proper programming skills. While such an approach works, it puts a barrier for a modeler, who wants to extend and customize the environment quickly according to his needs and possibly to experiment with several alternatives. The possibility to extend the modeling system via modeling would remove such barriers and enable modelers to create easily more productive modeling environments, which in turn would enable more productive MDD efforts using the customized environment. To enable model-driven plug-in development, it is necessary to provide an open modeling interface which we propose to call Application Modeling Interface (AMI). This interface would provide a set of model elements that the users can use for modeling their plug-ins. The plug-in model can then be loaded into modeling environment in order to enable custom features.

The paper goal is to introduce a conceptual framework for model-driven plug-in development of UML modeling systems. However, further AMI applications need to be implemented in UML based modeling systems in order to enable modelers to apply model-driven plug-in development.

In the next sections, we will review related works, present conceptual framework for model-driven plug-in development, and analyze three applications of AMI that have been implemented in MagicDraw modeling system and sample model-driven plug-ins built using them.

2. Related Works

In this section, we will discuss the works related to the topic of the paper. This paper introduces a novel approach, which to our knowledge was not concisely analyzed and described previously. However, it builds on the top of existing approaches of model driven development and extending systems via plug-in architecture and combines these two approaches to enable model-driven plug-in development for UML based modeling systems in particular.

Custom extensions of software applications are typically implemented as plug-ins, which architecture pattern is described by Fowler [13]. Some software systems are delivered as platforms with minimal functionality and most features are added by plug-ins developed based on that platform. Using plug-in architecture, existing applications can be extended without base modification to support new file formats, customer devices, or processing abilities [30]. The foundations of plug-in architecture were described in detail by Marquardt [30], who identified the major concepts of plug-in architecture and the relations among them. The state of the art of plug-in architecture implementation in software might be seen in Eclipse platform [6]. Eclipse plug-in implementation structure consists of a plug-in implementation library and plug-in contract defined as plugin.xml file. The plug-in implementation logic and its contract definition are essential for model-driven plug-ins architecture as well.

Model-driven development (MDD) is a software engineering approach consisting of the application of models and model technologies seeking to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying (making easier) and formalizing (standardizing, so that automation is possible) the various activities and tasks that comprise the software life cycle [20].

Current research shows that the model-driven development is successfully applied in context of various architectures, implementation platforms and software development processes. MDD has been used in mobile application development for specifying and generating user interface components and navigation schemas [10, 8], for developing interactive dynamic web applications [4, 5,15], for developing generic graphical user interface [31], automated user interface [43, 44] and system behavior testing [2]. MDD is successfully applied for developing service-oriented systems [47], embedded systems [34], distributed systems [1] and real-time systems [24] as well. Moreover, there are already applications of MDD where models are used as configuration of model-based software system extensions. Metamodels are widely used for configuring domain specific engines and environments such as model visual or textual editors [11, 27, 37, 38, 42]. For instance, automatic text completion feature in text editors of openarchitectureware [9] provides elements and their properties as described in EMF based metamodels [39]. The modeling environment of MetaEdit+ modeling system also follows the modeler defined language definition stored as a metamodel and automatically provides modeler with full modeling functionality: diagramming editors, browsers, and code generators [42].

Metamodels are also successfully used for specifying interfaces for model interchange among different modeling systems. The input and output data type of so called model bridges or model buses is elegantly configured by metamodels [1, 3, 7, 22, 23]. There are also successful cases when models are used to define model transformation data and transformation algorithm of modeling systems. Willink has proposed UMLX, a language for graphically specifying transformations [46]. The researchers from University of Paderborn have developed Fujaba Tool Suite, which can be used for specifying model transformation flow using transformation graph chart [45]. Although

Fujaba specification is not UML compliant, there is a successful Fujaba approach implementation using UML Activity diagram and AndroMDA code generation framework [35]. In this case, plug-in behavior is modeled using UML model and AndroMDA is used to generate plug-in executable code for specific UML modeling system.

UML-based modeling systems provide wide range of capabilities out-of-the-box in order to support different application domains and modeling approaches. However, typical modeler usually needs only a subset of capabilities provided by UML-based modeling systems. Moreover, modeler's required domain or modeling methods might not be supported by UML-based modeling systems at all. The vendors of UML-based modeling systems have solved these issues by providing capabilities customization and extension mechanisms for tailoring UML-based modeling system for modeler specific needs.

Nowadays, the most popular UML-based modeling systems such as IBM Rational Software Architect [26], MagicDraw [29], and Enterprise Architect [40] might be customized and extended using plug-ins based on system provided Application Programming Interfaces (API). However, UML modeling systems already successfully use models for their capabilities customization and extension. For instance, model patterns in IBM Rational Software Architect are defined using UML Collaboration elements, DSL engine of MagicDraw is configured by stereotyped UML classes, custom diagrams and their toolbars might be modeled as metaclass with attributes in Enterprise Architect. However, in most cases these UML-based modeling systems use models as a part of the functionality and are not considered as the main mechanism for capabilities extension or customization. The main mechanism for capabilities extension or customization in UML-based modeling systems is still a plug-in development based on system API.

3. Conceptual Framework for Model Driven Plug-in Development

In Figure 1, we define major concepts that are used in model-driven plug-in development and their relationships in order to establish a common vocabulary for the further use. This vocabulary can be treated as a conceptual metamodel that will be instantiated by different concrete applications.

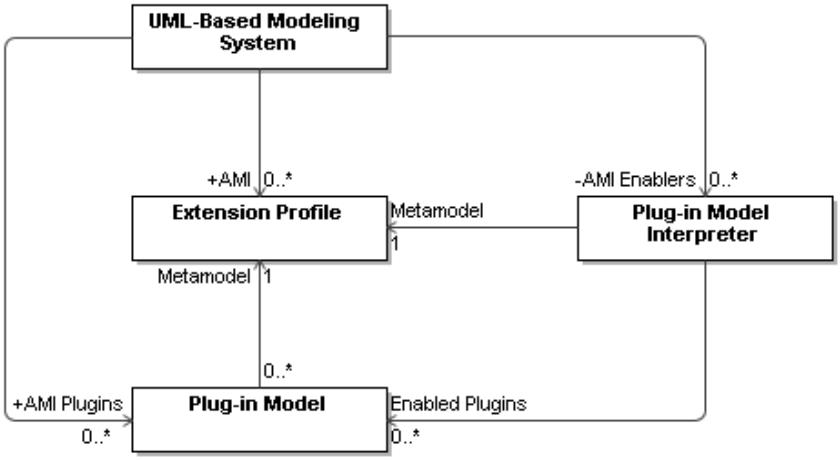

Figure 1. A conceptual metamodel for model-driven plug-in development in UML-based modeling systems

UML-based modeling system is a modeling environment, such as MagicDraw, IBM Rational Software Architect, and Enterprise Architect, which provides standard UML modeling capabilities. In order to support model-driven plug-in development, it needs to provide public access to a set of *extension profiles* that make up its Application Modeling Interface (AMI). Each *extension profile* should define a certain customization/extension point that is important for modelers, e.g. model validation rules, model transformation, model patterns, etc. The *extension profile* should be implemented using standard UML profiling capabilities (profiles, stereotypes, data types) that provide a necessary base for domain-specific language (DSL) definition [36]. A specific *UML-based modeling system* like MagicDraw may provide features for virtual transformation of UML profiles into first-class DSLs including important elements such as plug-in model completeness / correctness validation rules and custom diagrams enabling easier modeling [38]. When an extension profile is available as a part of AMI, a modeler can define *plug-in models* for specific modeling needs based on extension profiles. These *plug-in models* need to be executed by the *UML-based modeling system*. For achieving this, it should not only provide AMI realized as a set of *extension profiles*, but also provide a set of AMI enablers – *plug-in model interpreters*, one for each *extension profile*. In contrast to *extension profiles* that need to be directly used by the user, the *plug-in model interpreters* can be private internal features of *UML-based modeling system*. It is not necessary that all the available *plug-in models* are enabled at a particular time. Thus each *plug-in model interpreter* should keep

track of which *plug-in models* are enabled / loaded. *Plug-in model interpreters* should be implemented using the programming technologies used by the *UML-based modeling system* such as Java, C++ or .Net. In case an extensive open API is available, such *plug-in model interpreters* can be implemented as traditional code-driven plug-ins either by system vendors or users. Otherwise, they can be implemented as internal system features provided by system vendors. As *plug-in model interpreter* implementation is code-driven and dependent on a particular UML-based modeling system implementation technology and architecture, it is of no interest for further analysis in this paper. The *extension profiles* and *plug-in models* are defined based on standard UML capabilities, thus they are of interest in this paper.

In the next section, we will present three cases of model-driven plug-in development in *UML-based modeling system* MagicDraw focusing on *extension profile* and *plug-in model* definitions and illustrations how extension is enabled.

4. Model Driven Plug-in Development Applications in MagicDraw

In order to demonstrate the feasibility of the proposed conceptual framework, we have implemented in MagicDraw three cases of model-driven plug-in development capabilities: 1) *model validation*, providing capability to define custom rules for checking user model validation for completeness and correctness; 2) *methodology wizard*, providing capability to define custom step by step guidance for model content creation in form of wizard; 3) *model patterns*, enabling specification of model patterns and transformation for applying them in user models. These applications are very pragmatic (they are heavily used by MagicDraw users) and also rather different from each other. Therefore, they should serve as a good illustration of the proposed approach. In Figure 2, we present object diagram depicting these applications. We will focus on analysis of the object in the gray area, i.e. sample applications of model-driven plug-in development framework, as the upper part is product-specific and is out of scope of this paper.

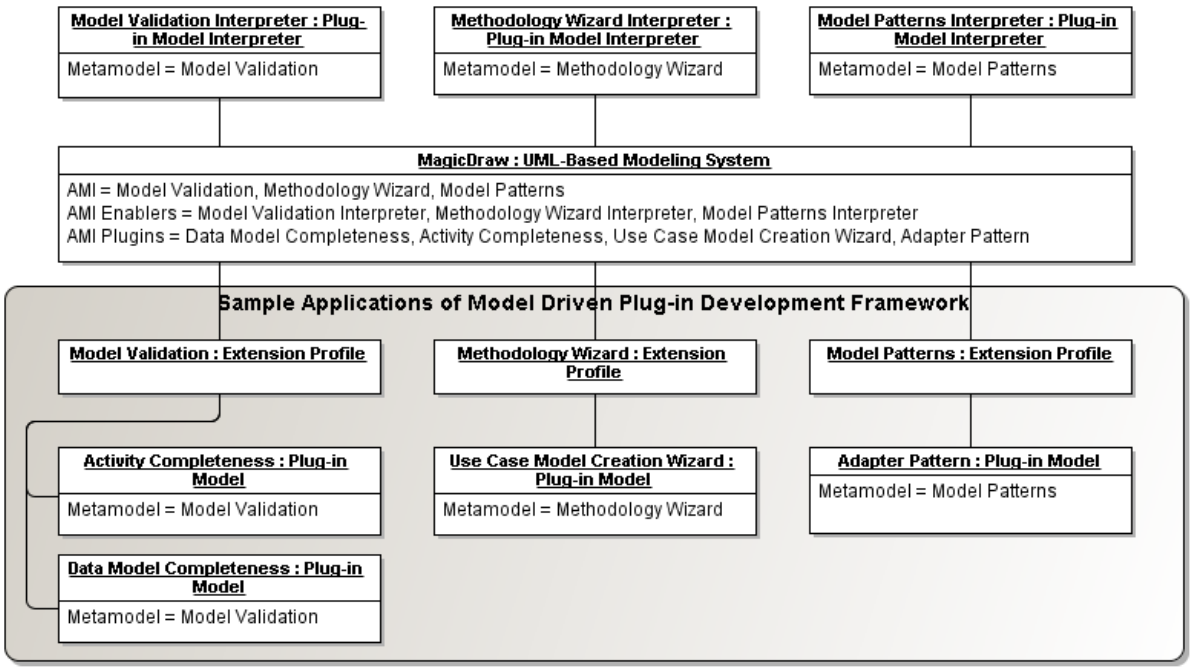

Figure 2. Sample applications of conceptual framework for model-driven plug-in development in UML-based modeling system MagicDraw

4.1. Model Validation

UML modeling systems typically enforce the rules defined in UML metamodel. However, the models can be incomplete or inconsistent according to rules that need to be followed in specific modeling methods. As UML is method-independent language, the UML modeling systems do not enforce those method-dependent rules, which are not standardized and vary between many different methods. Therefore it is necessary to allow users to define their own model validation rules.

Model validation rules might be assembled into suites that can be used to validate a model or some part of it. The validation suite may be active, which enables immediate validation while user is modeling. Each validation rule must define comprehensive message for explaining invalid situations. The severity of non-conformance may differ between validation rules.

4.1.1. Extension Profile for Model Validation

Validation rules are typically modeled as constraints in UML. The proposed extension profile for model validation capability extension modeling is

depicted in Figure 3. It extends Package and Constraint elements for specifying validation suites and rules in the model. Validation rules are modeled as a UML Constraint with «ValidationRule» stereotype applied.

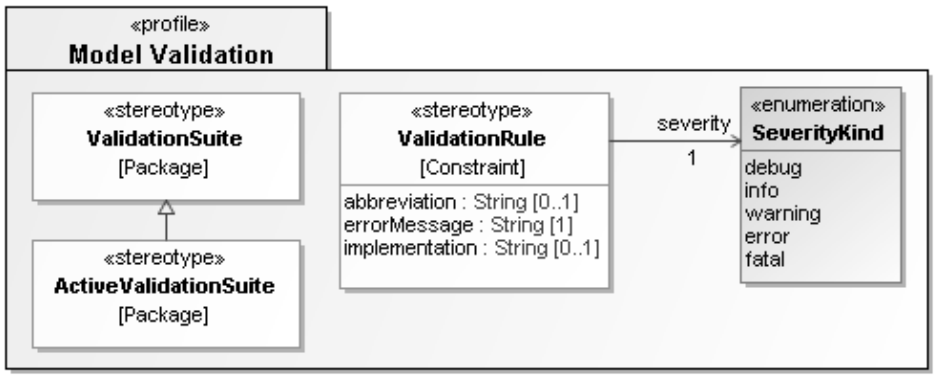

Figure 3. Proposed extension profile for validation rules capability extension modeling

Validation rule reuses Constraint properties *name*, *constrainedElement*, and *specification*. Validation rule *constrainedElement* property specifies model element for which this rule is applicable. When a UML metaclass is specified as a constrained element, the validation rule applies to all the UML elements that are instances of this metaclass. When a stereotype is specified as a constrained element, a validation rule applies to all the model elements that have this stereotype applied. When a classifier is specified as a constrained element, a validation rule applies to all the instance specifications of that classifier.

Validation rule specification property is used for specifying validation rule implementation expression. The expression might be written in Java [17] or in Object Constraint Language (OCL) [32].

The stereotype «ValidationRule» provides additional tag definitions *abbreviation*, *errorMessage* and *implementation*. The abbreviation tag is be used for specifying short name of the validation rule for grouping and filtering rules using validation user interface. The tag *errorMessage* defines the textual error message, which is displayed when invalid model element is found. The error message should explain the invalid situation and provide tips for solving it. The *implementation* tag is optional and used for specifying the Java class, which is used for validation rule implementation.

Validation rules might be composed into validation suites represented as packages with «ValidationSuite» or «ActiveValidationSuite» stereotype applied. Validation rules in simple validation suite are executed manually by the modeler on demand, however validation rules in active validation suite are executed automatically by the modeling system according to the model changes.

MagicDraw UML modeling system provides an Model Validation Interpreter (see Figure 2), which collects modeler created validation suites with validation rules from the models, parses their specifications and applies them for model elements validation.

4.1.2. Sample of Model Validation AMI Plug-in Model

Application of Validation Rules AMI Plug-in sample consists of the validation rule implementation for identifying “black hole” action in Activity diagrams. “Black hole” action is an action, which has no outgoing control flow relationships. Such an action does not specify the next action and stops the control flow at a dead-lock.

An OCL expression for checking if there is a control flow going out from an action is very simple:

context Action **inv** BlackHoleAction:

self.outgoing->size() > 0

The validation rule for “black hole” action is modeled as a constraint with «ValidationRule» stereotype applied and it is composed into a package *Activity Completeness* which represent the validation suite with stereotype «ValidationSuite» applied.

The OCL expression for checking action’s outgoing flows is specified in Constraint *Specification* property. The validation rule severity and error message are specified in additional properties added as tags of «ValidationRule» stereotype. When all of these properties are specified, modeler can validate his activities model according to the validation suite. A detection of a “black hole” action, which violates specified validation rule is depicted in Figure 4.

4.2. Methodology Wizards

Usually UML models are constructed visually drawing the elements in diagrams in an iterative manner. However, novice modelers often prefer to create model content using step-by-step wizard, which guides their modeling actions according a specific modeling methodology.

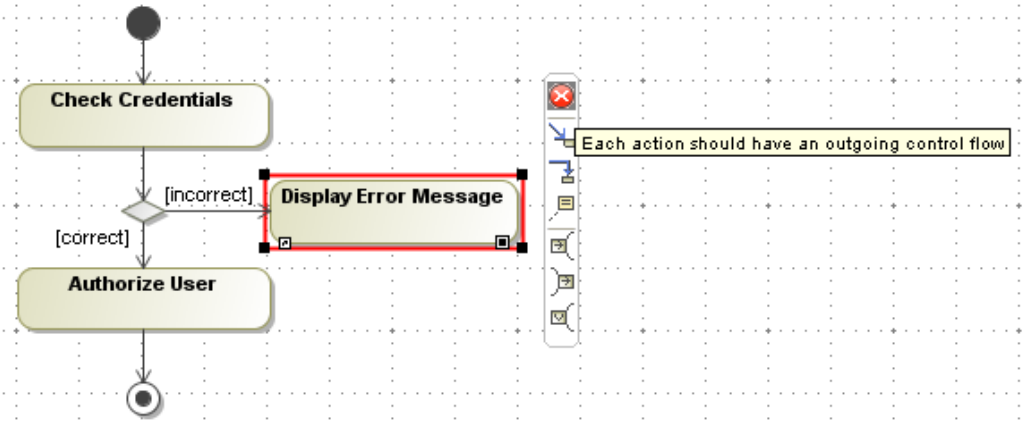

Figure 4. A detected invalid model element – a "black hole" action

4.2.1. Extension Profile for Methodology Wizard

The AMI for Model Based Methodology Wizards provides the capability for preparing step-by-step modeling wizards in model driven way. Modeler may prepare required wizard by modeling it using special type of activity, which stereotyped actions represent the steps of the wizard.

The proposed extension profile for AMI plug-ins of methodology wizards modeling is shown in Figure 5. It provides four stereotypes for specifying possible wizard steps: for specifying root element name step («SpecifyNameStep»), providing element description step («ProvideDescriptionStep»), capture and relate elements steps («CaptureElementStep» and «RelateElementStep»). The stereotype «Wizard» specifies the activity itself, which represents the methodology wizard.

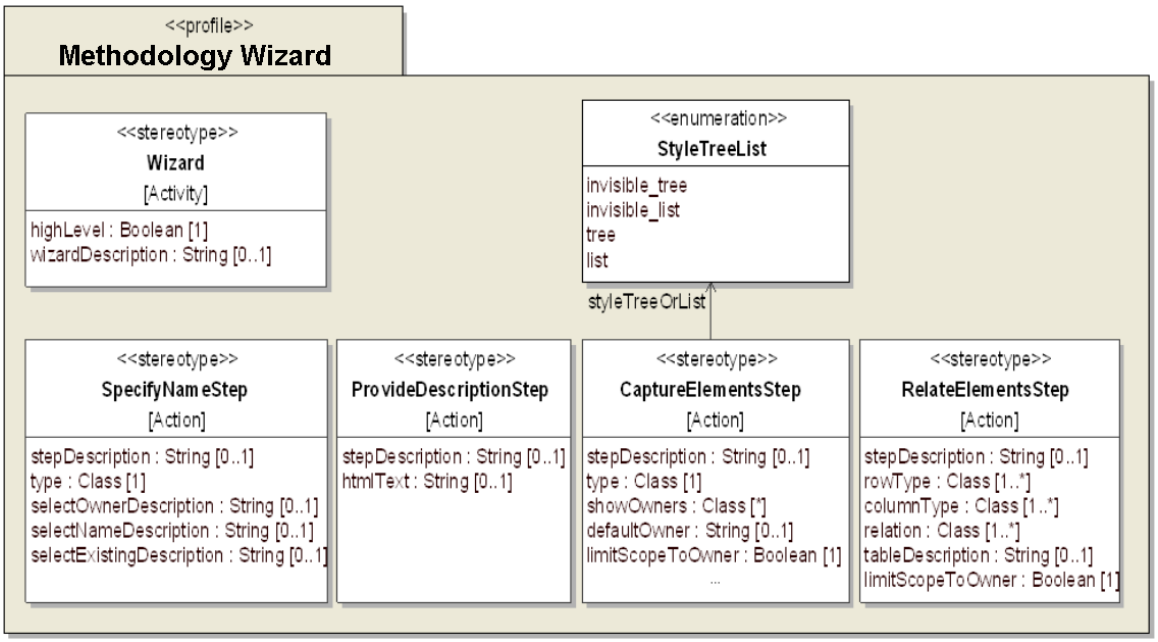

Figure 5. Proposed extension profile for Methodology Wizard capability extension modeling

Each step type requires additional information for specifying how to enable user's input and how the model should be constructed regarding given user's input. This information is modeled using step stereotype tags. Wizard step stereotypes contain tags for specifying various step configuration properties and model element types available during the particular step.

4.2.2. Sample of Methodology Wizard AMI Plug-in Model

The following sample represents the implementation of the Methodology wizard AMI plug-in for step-by-step Use Case model creation. The Use Case model creation process consists of modeling system actors, use cases, and the relationships among them. The sample AMI plug-in model of methodology wizard for Use Case model creation and the graphical

wizard enabled by the methodology wizards model interpreter are presented in Figure 6.

The sample wizard contains seven steps modeled as stereotyped activity actions. The order of wizard steps is modeled by control flow between actions. The first and the last steps are specified by the initial and final nodes.

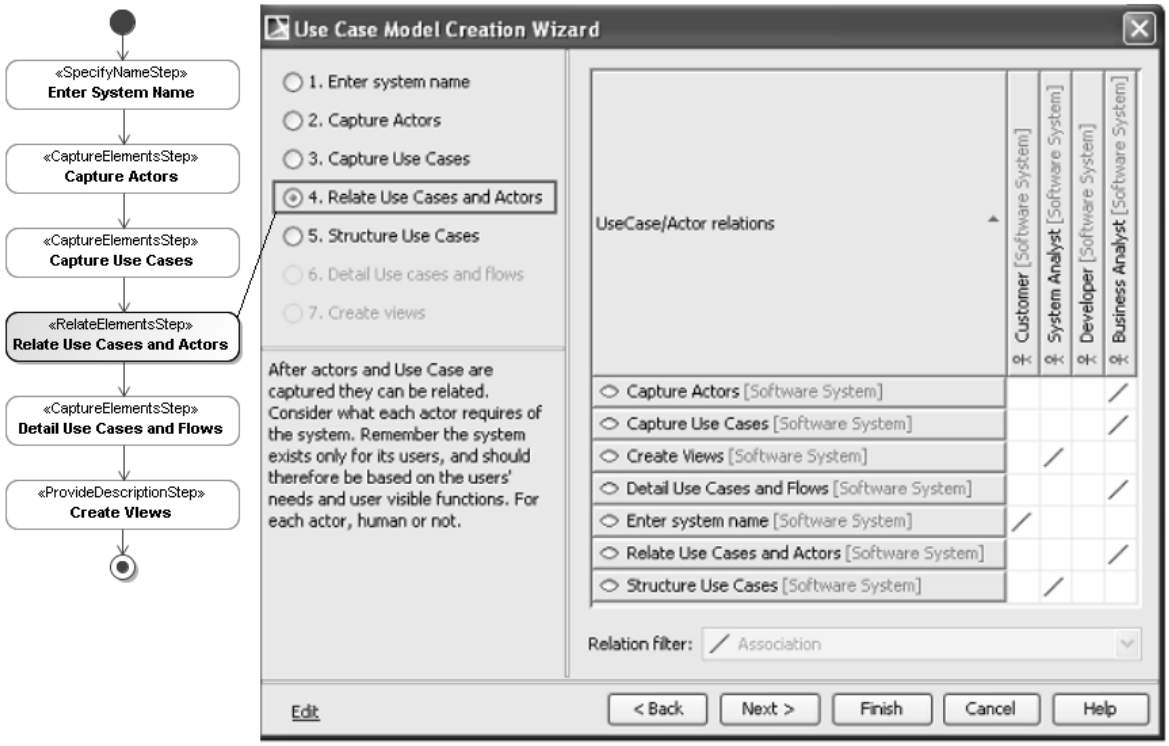

Figure 6. The Activity of the wizard for Use Case model creation and a screenshot of enabled wizard in action

4.3. Model Patterns

UML modeling systems provide implementation for various Java, JUnit, CORBA IDL, and XML Schema design patterns. A pattern may be applied on the target classifier for adding predefined classes and relationships between them. It may also visualize the created elements in the target diagram. The patterns functionality in UML modeling systems are typically implemented as a separate plug-in based on modeling system API. User may create new patterns and edit existing ones by using modeling system API. It is desirable to transform the current code-based approach for creating custom model patterns into a model-based one. Patterns structure, description and required model elements may be defined in a model.

There are many cases of model based pattern definitions already [28, 19, 14]. Most of them propose to use metamodels for pattern definitions [12, 25]. However, meta-modeling is an advanced technique and is not convenient for regular users as it does not represent concrete pattern elements but the metadata of its types. Another flexible approach for patterns definition modeling is based on UML collaboration element usage [41. Collaboration element represents a pattern itself, while collaboration role types represent the elements involved into the pattern. However, collaborations roles are limited by UML and might be used to represent classifiers only. For instance, UML packages cannot be presented in pattern using collaborations. The proposed AMI solution for model-based pattern definitions is defined using stereotypes and is not limited as collaboration based patterns. It also uses concrete elements of the model and does not require any meta-modeling skills. Moreover, it provides a diagram for pattern elements representation and visual layout specification. The elements layout presented in pattern diagram specifies the layout of pattern elements in target diagram where pattern is applied.

4.3.1. Extension Profile for Model Patterns

The proposed extension profile for AMI pattern plug-ins modeling is presented in Figure 7.

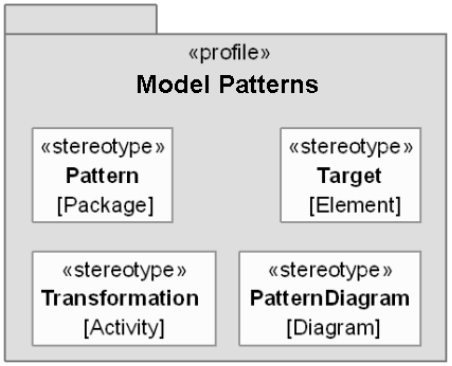

Figure 7. AMI Profile for Pattern modeling

The pattern is modeled as a UML package with «Pattern» stereotype applied. Pattern package is used to keep elements that should be created in the model after applying pattern. The «Target» stereotype specifies the element on which pattern should be applied. The layout of the pattern elements next to the target element is defined in the diagram with «PatternDiagram» stereotype applied. The pattern textual description is modeled as a comment of a pattern package.

4.3.2. Sample of Model Patterns AMI Plug-in Model

The AMI pattern model for Adapter pattern [16] is displayed in Figure 8. Adapter pattern classes, their relationships and visual layout are modeled in class diagram with «PatternDiagram» applied.

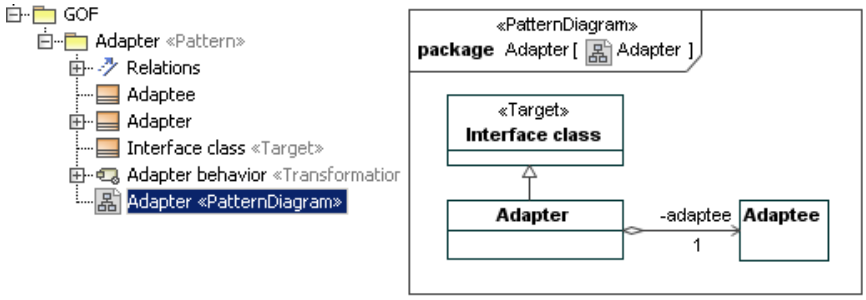

Figure 8. Adapter pattern implemented as AMI plug-in

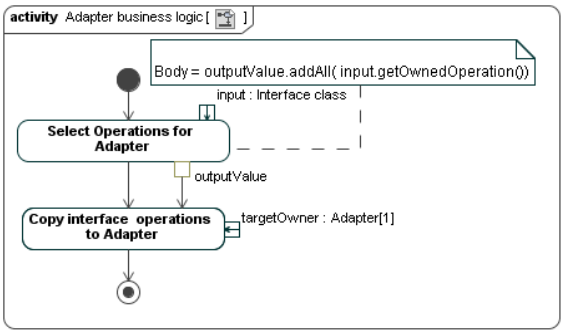

Figure 9. Adapter pattern business logic

The implementation of pattern business logic is modeled using activity diagram specified by the «Transformation» stereotype. The activity represents the pattern business logic as the flow of opaque actions. The body of opaque action contains executable code that actually implements the business logic of the pattern. The business logic of Adapter pattern is displayed in Figure .

The Adapter pattern business logic is described by two opaque actions. The first action selects all operations from the target class and pass them to the next action that actually copies selected operation to the Adapter class. The interpreter of pattern AMI plug-in interprets the pattern model and enables graphical user interface of the Pattern wizard (see Figure 10) for selecting and applying pattern.

Pattern description text presented in Pattern Wizard window is modeled using Comment element of pattern package. Pattern category tree in Pattern Wizard window reflects the structure of the pattern package and its parent packages in the model. Further steps of pattern wizard might be modeled as the actions of the transformation activity similar to the Methodology Wizards AMI described in previous Methodology Wizards section.

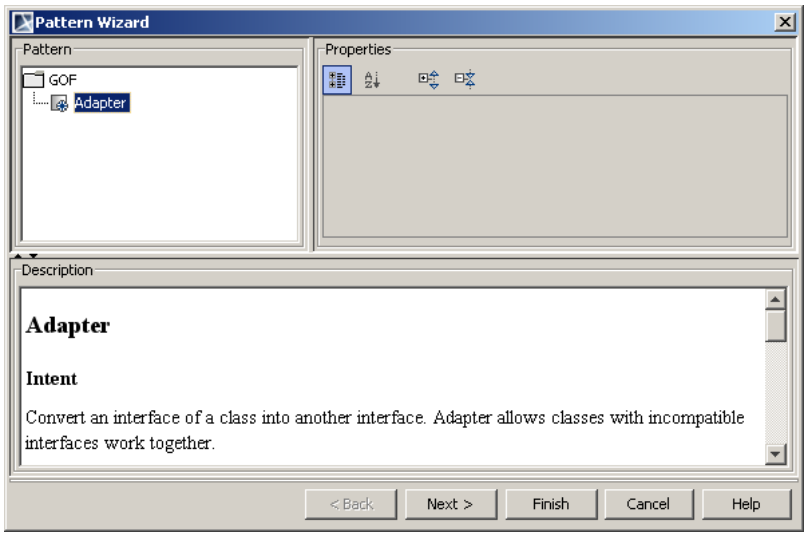

Figure 10. Pattern Application Wizard enabled by the interpreter with modeled Adapter pattern

5. Benefits and Drawbacks of Model-Driven Plug-in Development in UML-Based Modeling Systems

The proposed model-driven plug-ins development approach based on AMI might be compared to the plug-ins development using traditional programming using API. Since the AMI is a higher model-based abstraction of API, its benefits and drawbacks naturally come out of model-driven development practices described by Hailpern and Tarr [20].

The plug-in development using AMI should raise the abstraction of development tasks to the modeling level that enables modelers to extend modeling system without programming skills. Moreover, modelers may develop AMI plug-ins by modeling them in modeler-friendly environment of UML-based modeling system without a need to prepare and learn integrated development environments for specific programming languages.

While experimenting with sample applications of model-driven plug-in development in MagicDraw, we have measured and compared time required for developing the same functionality plug-ins using AMI and API based approaches. The same AMI and API based versions of model validation and model pattern plug-ins were developed in MagicDraw modeling system. At this particular set of plug-ins developed in MagicDraw, the time required for AMI based plug-ins development was shorter by ~60% comparing to time required to develop similar API based plug-in. However, from this simple observation we cannot claim that the productivity of developing AMI based plug-ins should be so high because there are many factors, such as complexity of UML-based modeling system API, modeling and programming skills, modeling environment usability, plug-ins debugging, and later plug-in maintenance, which can influence the results and should be taken into account in case of scientific comparison that aims to provide objective measures. The comparison of plug-ins development time mentioned here only indicates a faster development of particular plug-ins in model-driven way within a specific context of particular developers (authors of the paper), particular UML-based modeling system (MagicDraw), and particular integrated development environment (Eclipse). Statistical research experiments still need to be performed in order to get robust improvement estimates.

While model-driven plug-in development has multiple benefits they do not come for free – it also creates a few drawbacks, such as:

- The exposed extension profiles enable defining plug-in models in a higher abstraction level compared to code based plug-ins, which makes it easier to develop but limits the flexibility and possible extension variations;
- Runtime interpretation of plug-in models is typically slower compared to executing binary code-driven plug-ins. However, this issue might be solved by replacing plug-in model interpreter with code generator, which would generate executable plug-in code out of the plug-in model as proposed by Schippers et. al [35];
- Due to limitations of state-of-the-art UML-based modeling systems, it is more difficult to debug plug-in models compared to traditional code-driven plug-ins, which limits capabilities for building and maintaining complex plug-in models;
- Changing an extension profile requires changing plug-in model interpreters, thus each change in extension profile involves both modeling and programming technologies.

While the mentioned drawbacks are not critical and should not be show-stoppers for adopting this approach, it is important to take into consideration that the overall maturity of modeling practices and skills is much lower compared to programming and some disappointments about the proposed approach may come from abuse of modeling and UML in particular when designing extension profiles and creating plug-in models based on them.

It is also worth mentioning that model-driven plug-in development is already heavily used by MagicDraw R&D team, which applies it for implementing new product capabilities, and MagicDraw consultants and solution architects, who implement custom modeling solutions according to the custom needs of MagicDraw customers. This indicates that this approach is very pragmatic and is appreciated by the vendor itself.

6. Conclusions

The paper presented a novel approach to developing plug-ins in UML-based modeling systems taking into use model-driven development paradigm. The conceptual framework for practicing this approach was presented and illustrated by three different cases that were implemented in MagicDraw modeling system. From the work presented, we can draw the following conclusions:

- Model-driven development has significant advantages compared to traditional programming when applied to developing plug-ins for UML based modeling systems:
  - It enables modelers to develop custom capabilities applying their UML modeling skills and does not require knowledge, skills and tools for programming in languages like Java or C++;
  - It enables shorter plug-in development cycle because plug-in models can be built and directly interpreted inside the same UML modeling system as opposed to code based plug-ins that are typically developed in different development environments and require reboot of UML modeling system in order to enable the new functionality.

- Analysis of related works reveals that model-driven development is successfully applied in many domains, but the state of the art UML based modeling systems provide traditional plug-in development capabilities that are based on programming and support only fragments of model-driven approach;
- In order to enable model-driven plug-in development, UML modeling systems need to provide UML-based alternative to Application Programming Interface (API), which we propose to call Application Modeling Interface (AMI). AMI should be exposed as a set of extension profiles that provide capabilities for developing plug-in models that can be executed by internal plug-in model interpreters;
- Sample AMI implementations for supporting custom model validation, methodology wizards, and model patterns plug-ins demonstrate that this approach is feasible;
- While there are a number of practical arguments why model-driven approach for plug-in development in UML modeling systems is beneficial, a further research needs to be conducted in order to get statistical evidence on productivity improvements comparing it to a traditional programming-based approach.

In the future, we plan to expand AMI usage in MagicDraw for other extension cases, and conduct additional research to get a more detailed comparison of model-driven vs. code-driven plug-in development. Also, it is necessary to investigate how the same approach could be adopted in other UML-based systems as well as in other types of software systems.

References

[1] **J.P.A. Almeida.** Model-Driven Design of Distributed Applications. *CTIT Ph.D.-Thesis Series, No.* 06-85, *Telematica Institute Fundamental Research Series, No*. 018, 2006.

[2] **D. Barisas, E. Bareiša.** A Software Testing Approach Based on Behavioral UML Models. *Information Technology And Control, Kaunas, Technologija*, 2009, *Vol.* 38, *No*. 2, 119–124.

[3] **X. Blanc, M.-P. Gervais, P. Sriplakich.** Model Bus. Towards the Interoperability of Modelling Tools. *Proceeding of the European workshop on Model Driven Architecture: Foundations and Applications* (*MDAFA*' 2004), *June* 2004, *Linköping University, Sweden, selected for : Lecture Notes in Computer Science (LNCS) «Model Driven Architecture: Revised Selected. Papers», Vol*. 3599/2005, *Springer*. 17-32.

[4] **M. Brambilla, S. Ceri, F.M. Facca, I. Celino, D. Cerizza, E.D. Valle.** Model-driven design and development of semantic Web service applications. *ACM Trans. Internet Technol. Vol*. 8, *No*. 1, 2007, *Article* 3.

[5] **S. Ceri, F. Daniel, M. Matera, F.M. Facca**. Model-driven development of context-aware Web applications. *ACM Trans. Internet Technol. Vol*. 7, *No*. 1, 2007, *Article* 2.

[6] **E. Clayberg, D. Rubel.** Eclipse Plug-ins. *Third Edition. Massachusetts. Addison Wesley Professional*. 2009.

[7] **K. Duddy, A. Gerber, K. Raymond.** Eclipse Modelling Framework (EMF) import/export from MOF/JMI. *Technical report, CRC for Enterprise Distributed Systems Technology* (*DSTC*), 2003.

[8] **J. Dunkel, R. Bruns.** Model-Driven Architecture for Mobile Applications. *Proceedings of the* 10*th International Conference on Business Information Systems (BIS), Vol*. 4439/2007, 2007, 464–477.

[9] **S. Efftinge, P. Friese, A. Haase, C. Kadura, B. Kolb, D. Moroff, K. Thoms, and M. Völter.** OpenArchitectureWare User Guide, Version 4.3, 2008.

[10] **J. Eisenstein, J. Vanderdonckt, A. Puerta.** Applying model-based techniques to the development of UIs for mobile computers. *Proceedings of the 6th international Conference on intelligent User interfaces. IUI* '01, *ACM, New York, NY*, 2001, 69–76.

[11] **K. Ehrig, C. Ermel, S. Hänsgen, G. Taentzer.** Generation of visual editors as eclipse plug-ins. *Proceedings of the* 20*th IEEE/ACM international Conference on Automated Software Engineering, ASE* '05. *ACM, New York, NY*, 2005, 134–143.

[12] **M. Elaasar, L. C. Briand, Y. Labiche.** A Metamodeling Approach to Pattern Specification. *In MoDELS, Vol*. 4199/2006 *of Lecture Notes in Computer Science*, 2006, 484–498.

[13] **M. Fowler.** Patterns of Enterprise Application Architecture. *Addison-Wesley*, 2003.

[14] **R.B. France, D.-K. Kim, S. Ghosh, E. Song.** A UML-Based Pattern Specification Technique. *IEEE Trans. Software Eng., Vol*. 30, *No*. 3, 2004, 193–206.

[15] **P. Fraternali, P. Paolini.** Model-driven development of Web applications: the AutoWeb system. *ACM Trans. Inf. Syst. Vol*. 18, *No*. 4, 2000, 323–382.

[16] **E. Gamma, R. Helm, R. Johnson, J. Vlissides.** Design Patterns: Elements of Reusable Object-Oriented Software. *Addison-Wesley*, 1995.

[17] **J. Gosling, B. Joy, G.L. Steele.** The Java™ Language Specification. *Addison-Wesley*, 1996.

[18] **G. Goth.** Beware the March of this IDE: Eclipse is overshadowing other tool technologies. *IEEE Software, Vol*. 22, *No*. 4, 2005, 108–111.

[19] **A.L. Guennec, G. Sunye, J. Jezequel.** Precise Modeling of Design Patterns. *Proceedings of the* 3*rd International Conference on the Unified Modeling Language (UML)*, 2000, 482–496.

[20] **B. Hailpern, P. Tarr.** Model-driven development: the good, the bad, and the ugly. *IBM Syst. J. Vol*. 45, *No*. 3, 2006, 451– 461.

[21] **W. Harrison.** Eating Your Own Dog Food. *IEEE Software, Vol*. 23, *No*. 3, 2006, 5–7.

[22] **C. Hein, T. Ritter, M. Wagner.** Model-driven tool integration with modelbus. *Workshop Future Trends of Model-Driven Development*, 2009.

[23] **H. Kern, S. Kuhne.** Model Interchange between ARIS and Eclipse EMF. 7*th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA* 2007, 2007.

[24] **M.U. Khan, K. Geihs, F. Gutbrodt, P. Gohner, R. Trauter.** Model-driven development of real-time systems with UML 2.0 and C. *Model-Based Development of Computer-Based Systems and Model-Based Metho-*

dologies for Pervasive and Embedded Software, MBD/ MOMPES 2006, *Intl. Workshop*, 2006, 10–42.

[25] D.-K. Kim, R. France, S. Ghosh, E. Song. A UML-Based Metamodeling Language to Specify Design Patterns. *Proceedings of the 2nd Workshop in Software Model Engineering*. San Francisco, *USA*, 2003.

[26] M. Kunal. Introducing IBM Rational Software Architect. *http://www-128.ibm.com/developerworks/ rational/library/05/kunal/*, 15 *February* 2005.

[27] I. Kurtev, J. Bézivin, F. Jouault, and P. Valduriez. Model-based DSL frameworks. *Companion to the* 21*st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (*OOPSLA* '06), *ACM, New York, NY, USA*, 2006, 602–616.

[28] A. Lauder, S. Kent. Precise Visual Specification of Design Patterns. *Proceedings of the* 12*th European Conference on Object-Oriented Programming*, 1998, 114–136.

[29] No Magic. MagicDraw features, *http://www.nomagic. com/magicdrawuml/features.htm.*

[30] K. Marquardt. Patterns for Plug-Ins. *In Manolescu, D., Voelter, M., and Noble, J. Pattern Languages of Program Design* 5, *Addison-Wesley Professional*, 2005, 301–317.

[31] Z. Obrenovic, D. Starcevic. Model-Driven Development of User Interfaces: Promises and Challenges. In *Computer as a Tool*, 2005, *EUROCON. The International Conference on*. 2, 2005, 1259–1262.

[32] OMG. Object Constraint Language, Version 2.2. *http://www.omg.org/spec/OCL/2.2/*, 2010.

[33] OMG. MDA Guide Version 1.0.1, *http://www.omg. org/docs/omg/03-06-01.pdf*, 2003.

[34] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio. A Model-driven Design Environment for Embedded Systems. *Proceedings of the* 43*rd annual Design Automation Conference*, 2006, 915–918.

[35] H. Schippers, P. Van Gorp, D. Janssens. Leveraging UML Profiles to generate Plugins from Visual Model Transformations. *Electronic Notes in Theoretical Computer Science*, *Vol*. 127, *No*. 3, 2005, 5–16.

[36] B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. 10*th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing* (*ISORC*'07), 2007, 2–9.

[37] S. Sen, B. Baudry, H. Vangheluwe. Domain-specific model editors with model completion. *Models in Software Engineering, Springer*, 2008, 259–270.

[38] D. Silingas, R. Vitiutinas, A. Armonas, L. Nemuraite. Domain-Specific Modeling Environment Based on UML Profiles. *In: Targamadze, A., Butleris, R., Butkiene, R. (eds.), Information Technologies*'2009, *Kaunas* 2009, 167–177.

[39] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. EMF: Eclipse Modeling Framework, Second Edition. *Addison-Wesley Professional, Longman, Amsterdam*, 2009.

[40] Sparx Systems. Enterprise Architect 8 Reviewer's Guide. 2010, *http://www.sparxsystems.com/downloads/ whitepapers/EAReviewersGuide.pdf.*

[41] G. Sunyé, A.L. Guennec, J-M. Jézéquel. Design Patterns Application in UML. *Proceedings of the 14th European Conference on Object-Oriented Programming*, 2000, 44–62.

[42] J. P. Tolvanen, M. Rossi. MetaEdit+: defining and using domain-specific modeling languages and code generators. *In: Companion of the* 18*th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (*OOPSLA* 2003), *Anaheim, CA, USA, ACM Press*, 2000, 92–93.

[43] A. Ušaniov, K. Motiejūnas. A Method for Automated Testing of Software Interface. *Information Technology and Control, Kaunas, Technologija*, 2011, *Vol*. 40, *No*. 2, 99–109.

[44] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, J. Kazmeier. Automation of GUI testing using a model-driven approach. *Proceedings of the international Workshop on Automation of Software Test, AST* '06, *ACM*, 2006, 9–14.

[45] R. Wagner. Developing Model Transformations with Fujaba. *In H. Giese and B. Westfechtel, Eds., Proceedings of the* 4*th International Fujaba Days, Bayreuth, Germany, Technical Report*, 2006, 79–82.

[46] E.D. Willink. UMLX: A graphical transformation language for MDA. *In A. Rensink (Ed.) Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications*, 2003, 13-24.

[47] X. Zhang. Model Driven Data Service Development. *In Networking, Sensing and Control, ICNSC, IEEE International Conference*, 2008, 1668–1673.

Received March 2011.