DOMAIN-SPECIFIC MODELING ENVIRONMENT BASED ON UML PROFILES*

Darius Silingas[1], Ruslanas Vitiutinas[2], Andrius Armonas[3], Lina Nemuraite[3]

[1] *No Magic Europe, Training Department, Savanoriu av. 363, LT-49425 Kaunas, Lithuania, dariuss@nomagic.com*

[2] *Vytautas Magnus University, Faculty of Informatics, Vileikos st. 8, LT-44404 Kaunas, Lithuania, r.vitiutinas@if.vdu.lt*

[3] *Kaunas University of Technology, Department of Information Systems, Studentu st. 50-308, LT-51368 Kaunas, Lithuania, {andrius.armonas, lina.nemuraite}@ktu.lt*

Abstract. Domain-Specific Modeling Languages (DSML) play a key role in model-driven development. There are many approaches how to create a DSML. Recent trends in domain-specific modeling languages and issues of creating and using UML profiles are discussed in this paper. Then we present a novel approach for defining a full-featured DSML based on a UML profile and its customization instead of heavyweight metamodeling. This approach was implemented in MagicDraw UML tool and already successfully accepted by its users. Also, MagicDraw UML developers themselves applied it for creating SysML, DoDAF and UPDM modeling environments. The main benefit of this approach is that it allows a DSML modeler to reuse powerful features of already existing tools. We propose a seven-step DSML development process and illustrate it by an example demonstrating creation of a DSML for modeling organization structures. We also discuss benefits of taking the presented approach and some ideas for future enhancements based on the feedback of users.

Keywords: DSML, UML, UML profile, MagicDraw UML.

1 Introduction

There were many modeling languages and notations 15-20 years ago. Most of them, like OOSE [15], OMT [26], Booch [2], CRC cards [32], and ER [5] were designed for certain domains. Then, the UML [23] language was created as a reconciliation of those (and other) methods resulting in a general-purpose software modeling language. Starting from UML 2.0 version, the language was targeted not only at software systems, but also at embedded, business, real-time, system modeling, and other domains. However, our practice shows that UML without modifications can only be used efficiently for modeling software systems. Therefore new approaches how specialized modeling languages can be easily defined are under active research. Some parallels with programming world can be observed here: in the recent years, such languages as Ruby [29] and XML [28] became very popular because of their ability to introduce domain-specific concepts and define new languages. In the modeling world, correspondingly, new modeling approaches are getting popular which also allow definition of new domain-specific languages. A domain-specific language (DSL) is usually a formal, interpretable language for describing a specific domain (or part of that domain) for which the system is being designed. In this paper, we mostly focus on creation of graphical DSLs designed for modeling purposes, which we call domain-specific modeling languages (DSML).

During the last decade a number of techniques and approaches evolved where DSMLs play the key role, namely MDE (Model Driven Engineering), MDD (Model Driven Development), and MDSD (Model Driven Software Development), MDA (Model Driven Architecture) [20], MDI (Model Driven Integration) [7], DDD (Domain Driven Design) [11], and Microsoft Software Factories [13]. Recently, OMG issued a number of domain-specific modeling languages: the SysML [22] language for systems modeling, MARTE [24] profile for real-time and embedded systems modeling and UML Profile for DoDAF/MODAF (UPDM) [25] for modeling defense systems. Besides software development, domain-specific modeling languages are capable to support many emerging fields of activities, particularly enterprise knowledge modeling [14], creation of rule repositories [17] etc. For implementing DSML, there are two fundamental approaches: the heavyweight one, creating a new language from scratch by using metamodeling techniques, or the lightweight one, reusing and extending existing environments by creating UML profiles. Both of them have their advantages and drawbacks.

In this paper, we present a novel approach how a new DSML can be created on top of a UML profile combining features from both extending UML and creating a new language from the ground up. It allows quickly designing and easily maintaining a new DSML, which is based on UML profile, but does not have

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

features irrelevant for its domain-specific purposes. Moreover, support for model transformations, model comparison and merge, validation, code generation, and other features supported by UML tool can be reused without modifications in DSML environments created using our approach. This approach has already been implemented in MagicDraw UML tool, tested and praised by its users. Among other applications, it was used by MagicDraw tool developers themselves for implementing SysML, UPDM and DoDAF modeling environments that are successfully used by our customers in practice.

The remainder of this paper is structured as follows: in section 2 we review related work in the field, mostly other DSML tools and techniques they are based on; in section 3 we shortly present the supporting DSML framework implemented in MagicDraw UML; in section 4, a workflow for defining a new DSML is illustrated by a case study on organization structure modeling; in section 5, user feedback is analyzed; in section 6 we summarize results of our research and draw conclusions.

2 Related Work

The development of a DSML involves definition of the syntax, semantics, validation, and transformations to models or code. We can distinguish between two approaches that are used in the field of development of domain specific modeling languages: create a DSML as an instance of a certain metamodel, or create a DSML by extending some modeling language. While the first approach deals with the composition of languages, the second is related to their extension. Currently many research papers are focusing on improving the metamodeling-based development of DSMLs. However, not much research is done on combining those two approaches.

Today metamodeling environments are usually based on metalanguages close to MOF: GME (Generic Metamodeling Environment) supports MetaGME [19]; MetaEdit+ supports GOPPRR (Graph-Object-Property-Port-Role-Relationship) [30], Eclipse EMF and oAW – Ecore [31], XMF-Mosaic – Xcore [6], and AMMA (ATLAS Model Management Architecture) supports KM3 [16]. Microsoft DSL tools are promoting concept of Software Factories [13] and use their proprietary language and notation. Such a variety of metamodeling languages causes problems related with interoperability of tools, produced artifacts and their reuse.

One of the first successful commercial tools in that area was MetaCase MetaEdit+ [30]. The tool allows creating domain-specific languages as separate metamodels. It also has a means to specify graphical representation for the DSML constructs as well as define semantics on the created metamodel which can be later used for validation purposes; and specifying a code generator, which would transform the user model into the code. A bit more popular framework called EMF is available in Eclipse environment. Together with GMF and GMT projects [9], it can be used for achieving analogous results as with the MetaEdit+ tool. Eclipse offers the Ecore metamodel which is actually a light version of MOF [21] for creating DSMLs, the GMF framework for creating graphical representation of DSL elements, and the GMT framework tools for model transformations. Microsoft DSL Tools is an alternative option to the tools mentioned above in a way it offers a heavyweight approach to DSL creation.

Abouzahra et al [1] have noticed the problems related with the volume of artifacts produced by using metalanguages as well as creating UML profiles. They emphasize the need for interoperability between these approaches and propose to use special weaving metamodel in their AMW tool for external linking of models. Interesting approach is suggested in [3] for interconnecting different DSLs by using ontologies. The authors created a common upper ontology for software modeling languages and employed it for integrating DSMLs. The Moses framework [10] has the ability to embed other languages. This addresses quite an important problem of combining or relating one DSML to another. In this case internal language is employed to do certain subtasks.

Bridging the gap of creating ordinary DSLs (textual languages) and DSMLs is also an important task. Some ideas on that are proposed in [12]. The author proposes to formulate languages as relational schemas, very similarly to what is done with DSMLs, thus eliminating the need for the parser as such. Such implementations naturally gain features like refactoring, versioning, or merging implemented easier because a unified repository might be available at the DSML developer hands.

Chen et al [4] see the greatest problem in the lack of a practical, effective method for the formal specification of DSML semantics. They propose a formal methodology based on Abstract State Machines with supporting GME toolset to anchor the semantics of DSMLs to precisely defined and validated “semantic units” that could lead toward an infrastructure for DSML design integrating formal verification and automatic generation of code translators with practical engineering tools.

Authors of [27] and [18] propose the use of UML profiles. However this approach does not allow creating a proper DSML, as after applying stereotypes on UML elements, these elements are still UML elements and preserve the semantics associated with them. UML profiles are designed to extend the language, but not to restrict it or convert it to another language. They are best applicable to situations where the DSML does not have a large deviation from the standard UML metamodel.

In contrast to both approaches described in this section, we propose a lightweight approach combining features from both worlds – from DSMLs based on metamodels and DSMLs based on UML profiles. By saying "lightweight" we mean that DSMLs built using our approach heavily reuse the UML infrastructure both at the language level and tool level and creators of those DSMLs usually do not have to use any programming at all. The proposed approach is implemented in MagicDraw UML tool. We will describe it in detail in the next sections.

3 MagicDraw UML Framework for Creating DSML

The key idea of our approach is to build a "customized UML profile", which means that we do not use only plain profiles to define a DSML. By definition a stereotype only extends an element of the UML language by adding additional properties. However, the semantics and appearance of the element remains unchanged. If we apply a stereotype to a class, the stereotyped element is still a class, which just has additional properties defined by the applied stereotype. We propose to add one more layer to profile modeling. We call this layer a "customization module". It contains customization classes that customize stereotypes by virtually transforming them to completely new metamodel elements. As seen in Figure 1, the customization is two-fold: on one hand it transforms stereotypes to new metamodel elements by defining customization classes for stereotypes; on the other hand it restricts the UML metamodel by hiding unnecessary language parts and enabling certain rules for relating UML elements to the new domain-specific elements.

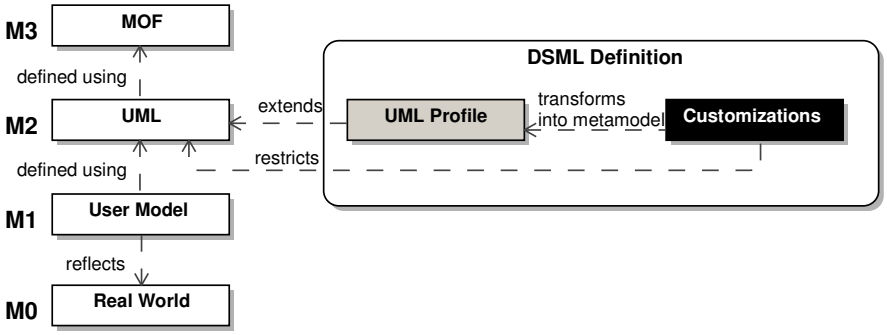

Figure 1. DSML definition framework in the context of OMG meta-layer architecture

Standard UML provides stereotypes, constraints and profiles for extending the UML language. We propose some enhancements of the standard UML profiling mechanisms enabling profile validation and customization. For validation, we add a stereotype `<<validationRule>>` for Object Constraint Language (OCL) constraints that should be used for validating the correctness and completeness of the domain-specific user model. In addition to the OCL constraint specification, this stereotype defines the following custom properties (tags):

- `severity` – importance of validation error (debug, info, warning, error, fatal);
- `error message` – a detailed explanation for displaying to the modeler if a model element does not conform to the constraint specification;
- `abbreviation` – a shorter reference name for identifying validation error type.

Model validation should be based on a collection of validation rules. For this purpose, we propose a stereotype `<<validationSuite>>`, which is applicable to `Package` metaclass instances. This enables grouping validation rules into suites. In a case a modeler needs to define a selection of validation rules from multiple packages, validation suite should use `import` relationships to the selected validation rule constraints.

We also propose to allow groups of tag definitions for structuring purposes, and custom dialog-based relationship style definition for creating on-the-fly scalable icon image for stereotyped relationships. While these are minor implementation-specific enhancements, we have found that they are highly demanded in industrial practice.

Although stereotypes allow specialization of UML metaclasses, they do not hide the fact that the first-class modeling elements are UML concepts, e.g. `Class`, `Actor`, `Package`, `Dependency`, with their properties. Since this is confusing and distracting for domain users, we propose to use a special mechanism, which virtually converts stereotypes to metamodel elements, i.e. stereotyped elements are treated as instances of new metaclasses in modeling environment. We propose a stereotype `<<Customization>>`, which contains tags for customization purposes, see Table 1.

Table 1. The main tags for stereotype *Customization*

Tag Name	Description
customizationTarget	The stereotype for which the customization applies
representationText	Alternative name to be used in the modeling environment instead of the UML element + stereotype names
usedUMLProperties	Standard properties of UML element to be used by the customized element
allowedRelationships	Relationships allowed/disallowed to connect to the customized element
disallowedRelationships	
typesForSource	Elements allowed to be used as source/target for the customized relationship
typesForTarget	
suggestedOwnedDiagrams	Diagrams/model elements that should be suggested to create in the context of the customized element
suggestedOwnedTypes	
possibleOwners	Possible owners of the customized element

When designing a DSML, we should create customizations for each stereotype. We suggest storing customizations in a separate module in order to separate the two parts of DSML (profile and customization). When the customization module is loaded into the modeling project, the modeling environment should find all model elements stereotyped by <<Customization>> and should reflect these settings appropriately in user interface components such as model element specification dialogs, model repository browser, contextual menus, report scripting language, etc. The specified restrictions for ownership and relationships should be enabled during creating model elements and drawing diagrams.

4 DSML Development Process

In this chapter, we propose a simple seven-step process for creating a DSML environment based on customized UML profile, which is described in the activity diagram presented in Figure 2. In the following subsections we will give a more detailed description for each of the tasks presented in Figure 2 together with simplified examples for building and using a DSML for organization structure modeling.

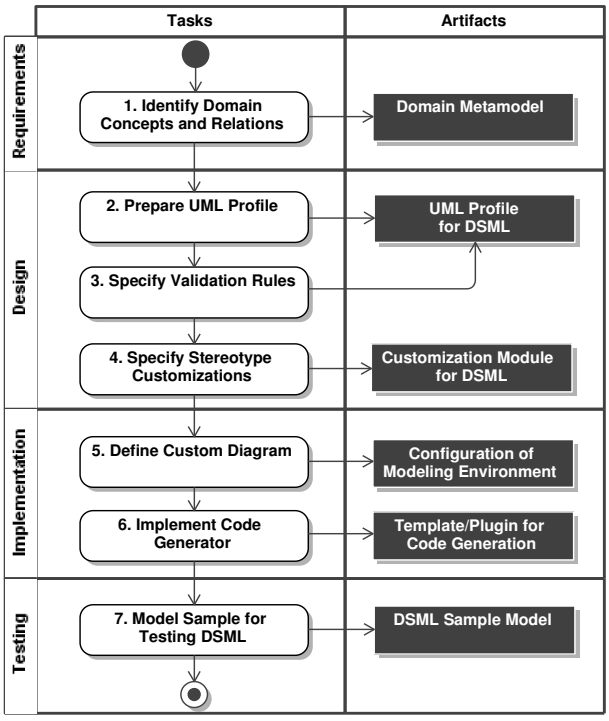

Figure 2. A process for creating DSML using customized UML profile

4.1 Identifying Domain Concepts and Relations

First, we have to identify domain concepts, their properties, and relations. For creating domain metamodel, it is possible to use a UML class diagram limited to classes, their properties, and associations (this subset of UML infrastructure is reused in both MOF and UML superstructure). In Figure 3 we present a sample metamodel for defining an organization structure. The properties of classes and association ends are hidden in order to minimize a complexity of the diagram.

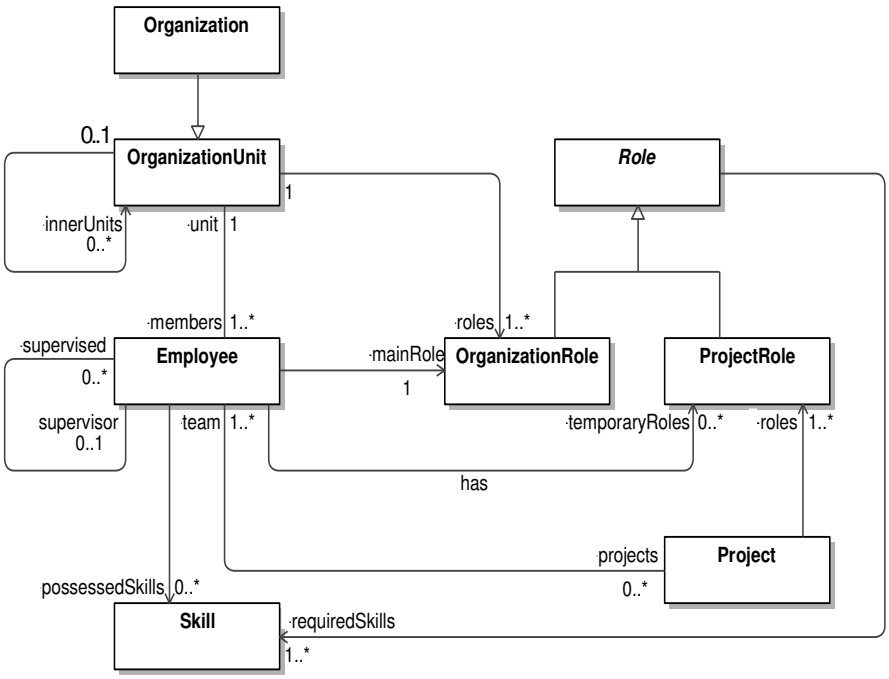

Figure 3. Conceptual metamodel for defining organization structures

4.2 Preparing a UML Profile

Since we are building DSML based on UML profiles, we need to map the metamodel to UML metaclasses and necessary extensions. The mapping of the sample organization structure metamodel to UML profile is presented in Figure 4.

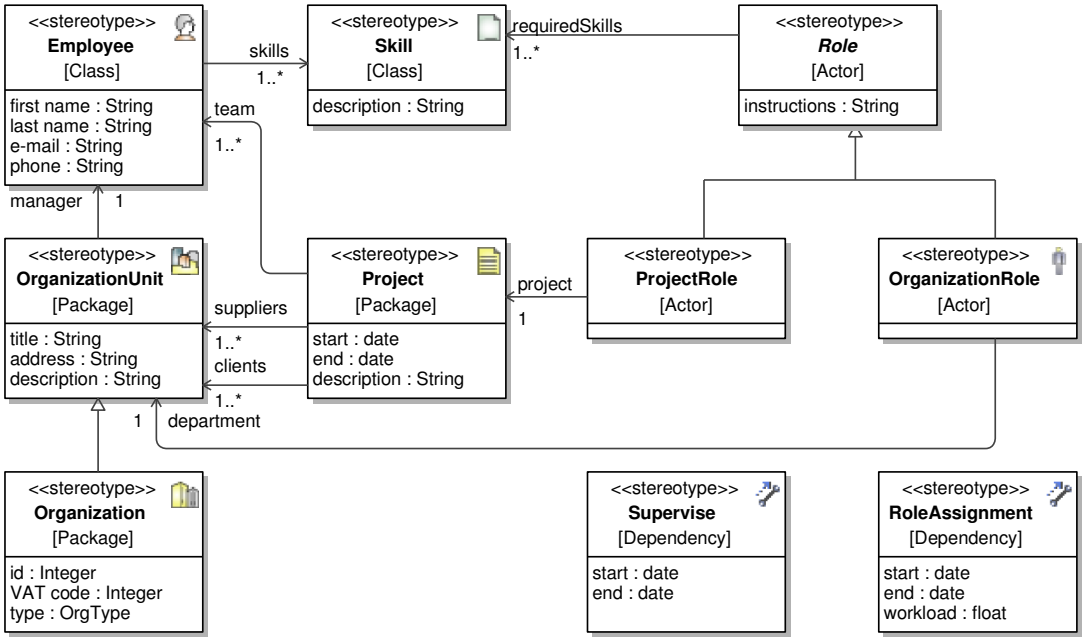

Figure 4. UML profile for defining organization structure

The mapping is a non-trivial task because the deep knowledge of how to apply the UML language is needed. Most of the concepts, e.g. `Organization`, will map to stereotypes on a selected metaclass, e.g. `Organization` stereotype on `Package` metaclass, with tags defining additional properties that are missing in UML metamodel, e.g. `ID`, `VAT code`, `type`. Some of the domain concept properties will map straightforwardly to UML metaclass properties, for example, the `name` property. Some of the relations defined in the metamodel will be mapped to standard UML metamodel relations, e.g. the relation `contains` from `Organization` to `OrganizationUnit` is mapped to a standard UML ability to contain a package inside another package. The other relations should be mapped to stereotypes, e.g. the relations between `Employee` and `OrganizationRole` or `ProjectRole` should be mapped to stereotype `RoleAssignment` on `Dependency` metaclass with additional tags defining `start` and `end` dates and `workload` ratio. Also we define icons for most of the stereotypes, including custom line styles and ends for path stereotypes. This allows the modeler to use intuitive symbols instead of UML shapes that might be unacceptable for the specific domain needs.

4.3 Defining Validation Rules

Some of the properties defined in metamodel will be difficult to map to stereotypes and their properties. Those ones should be mapped to UML constraints specified in OCL. Current state-of-the-art modeling tools have integrations with OCL execution environments like Dresden OCL2 toolkit [8]. These constraints can be used as validation rules to ensure correctness and completeness of DSML user models. The metamodel rule that an employee could be supervised by at most one supervisor can be expressed in this OCL constraint defined in the context of `Supervise` stereotype:

```
context Supervise inv singleSupervisor
Supervise::allInstances()->excluding(self)->
            forAll(s | not (s.supplier = self.supplier))
```

In Figure 5, we present a screenshot from MagicDraw UML modeling environment displaying a result of the validation of model not conforming to this validation rule.

We should define multiple validation rules for covering all the important aspects for model correctness and completeness. Most of the validation rules will be defined in the context of stereotypes. We should apply stereotype `<<validationSuite>>` to the profile itself to enable using all its validation rules for validating DSML user model.

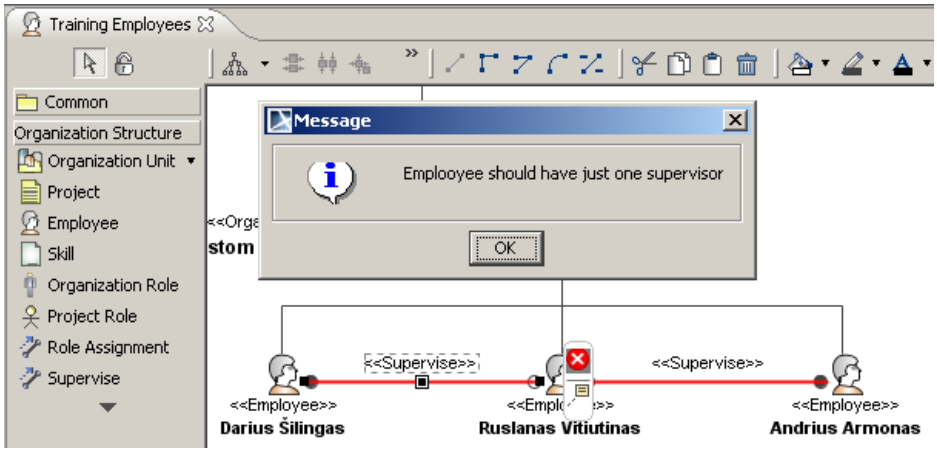

Figure 5. A screenshot indicating validation error of double supervising

4.4 Customizing the Language

For visualizing the possibilities of customization, we present a sample customization element and a resulting specification dialog for model element to which the customized stereotype was applied, see Figure 6. We should define such customizations for every stereotype in the organization structure profile.

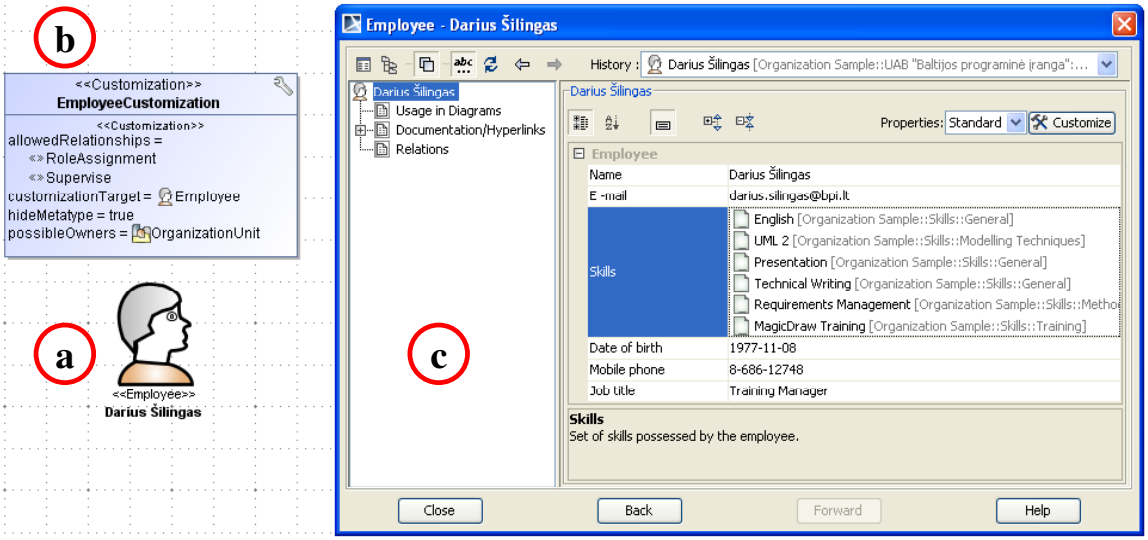

Figure 6. Diagram symbol representing user defined DSML element (employee); b) DSML customization element; c) a custom specification dialog for editing employee's properties.

4.5 Defining Custom Diagrams

For easy usage of a DSML, it is recommended to create a custom diagram that is limited only to domain modeling concepts. MagicDraw UML provides a wizard for this purpose, which takes user through the steps of specifying diagram name, type, and icon; selecting profile(s) and toolbars to use; constructing specific toolbars from DSML elements; specifying symbol properties and rules for smart manipulators (quick access relationships and elements). A sample screenshot of the resulting custom diagram was presented in Figure 5. Note that the diagram contains customized organization structure modeling toolbar containing selection buttons for defined DSML concepts. Together with stereotype customizations, the custom diagram provides the simple and intuitive DSML environment hiding the complexity of UML, but reusing the powerful features of modeling environment of MagicDraw. The created diagram would be portable to other modeling environments by using XML for saving custom diagram setup and exporting/importing diagrams in the same tool across different user machines.

4.6 Implementing Code Generators

It is a common practice to use UML models as an input for code generators and MDA tools to generate implementation-level artifacts such as source code, XML or database schemas. In most tools, generation of the code is based on textual templates, where the specific template engine parses those templates and fills-in values from the model.

The DSML environment should provide ability to use domain specific types and their properties in templates in the same way as standard UML classes and properties are used. We have adopted the *Velocity Template Language* (VTL) for creating model report engine and implemented the ability to explicitly reference DSML elements and their properties in VTL specifications (as opposed to accessing their info through UML metaclasses and stereotype properties). This makes it possible to write code generators that use only domain concepts to generate the code.

As an example of using the proposed template engine we will generate organization structure web site from the user model produced in previous sections. This task might be performed by creating a HTML page filled with dummy data in any HTML editor and replacing dummy data with references to model elements and properties. The Figure 7 shows a code fragment for representing employee's info in a HTML template (note that domain-specific concepts are explicitly used for code generation) and the Figure 8 shows the outlook of the generated HTML page in Web browser.

In VTL scripts, employee's personal information is accessed using the `firstName`, `lastName`, and `email` properties of `Employee`. Employee's organization unit is obtained from standard UML property `owner` as the `Organization Unit` is a customized `UML Package` element. A standard *Velocity* `#foreach` directive is used for iterating over employee skills referenced from `skills` property of `Employee`. Using this report engine, we may generate a full-featured web site representing complete organization structure.

```
<table>
   <tr>
      <td>Full Name</td>
      <td>$employee.firstName $employee.lastName</td>
   </tr>
   <tr>
      <td>Organization Unit</td>
      <td>$employee.owner</td>
   </tr>
   <tr>
      <td>E-Mail</td>
      <td><a href="mailto:$employee.eMail">$employee.eMail</a></td>
   </tr>
   <tr>
     <td>Skills</td>
      #foreach ($skill in $employee.skills)
        <tr><td>$skill.name</td></tr>
      #end
   </tr>
</table>
```

Figure 7. A screenshot of HTML template with VTL scripts for generating employee info pages

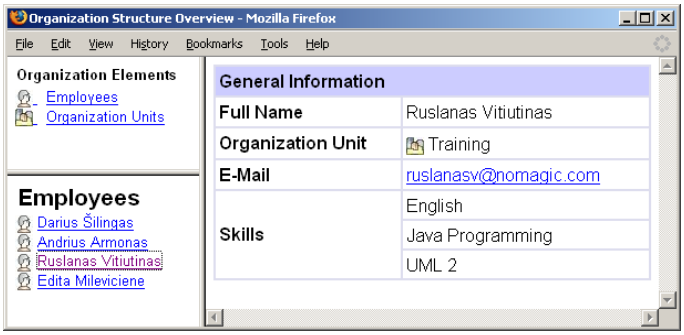

Figure 8. A screenshot displaying how the generated HTML code looks in a browser

In the same way, we can make templates for generating relational database or XML schemas from the organization structure metamodel. More sophisticated code generation features are available in MDA tools that typically use UML models as input. Since we use DSML built on the top of UML profiles, those tools can easily be reused with organization structure models as well.

4.7 Testing DSML and User Models

It is very important to set the rules for validation of user models. For example, for modeling organization structure we can set the following rules:

- The top-level element should be an `Organization`;
- An `Organization Unit` should be defined inside the `Organization` and can possibly be nested under the higher level `Organization Unit`;
- Inside each `Organization Unit` we should have packages `Projects` and `Roles` that are used for storing appropriate model elements;
- `Skills` should be defined in a separate package structure outside of the `Organization`;
- An `Employee` should be defined inside the `Organization Unit`, for which he primarily works.

For testing these rules the user should create a sample model containing all DSML elements in valid and invalid modeling situations for testing each validation rule. Although we present this as the last step, in practice it might be created and developed iteratively. Many modelers will prefer test-driven approach when a small model sample would be added before defining each validation rule.

A fragment of the organization structure model repository is shown in Figure 9. It is also a recommended practice to define a sample model project, including the profile and defining empty model structure that could be used as a template for starting a new model.

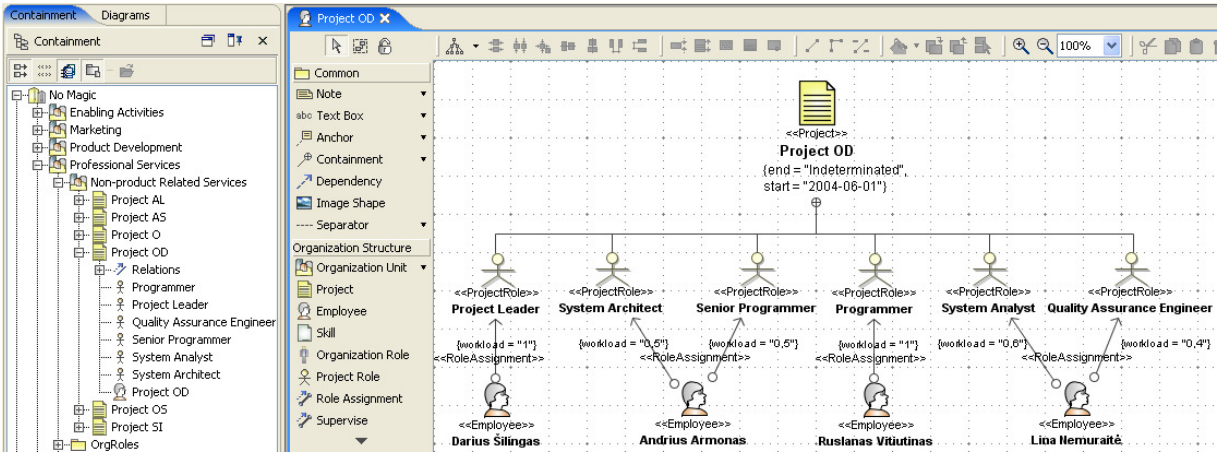

Figure 9. A screenshot displaying a fragment of the organization structure model repository and a DSML diagram specifying project roles and assigning them to organization employees

5 User Feedback and Future Work

Being members of MagicDraw UML R&D team, we have got a lot of positive feedback from our users after implementing and releasing the DSML engine in our tool. Also, our request database shows 163 requests for improvements proposed by our users. We cannot extensively analyze all of these requests due to space restrictions in this paper but we can clearly observe general trends where the suggested DSML approach needs to be improved. We will briefly discuss the main issues reported by our users that we are working on currently.

The first and the biggest group of issues is the lack of functionality to change the appearance of standard UML symbols when designing a new DSML. It is currently possible to change the icon of the symbol, add or remove additional compartments, hide existing subsymbols such as names, tags, specify which symbols can by connected to this symbol, etc. However, this approach is insufficient if it is needed to create a completely new symbol, which has nothing in common with existing UML symbols. OMG is currently working on the Diagram Definition Metamodel definition. This metamodel allows specifying graphical symbols. After implementing this metamodel in our tool, users will be able to design their own symbols that are completely different from UML symbols.

The other issue is that current DSML engine does not allow changing default UML metamodel values. What that practically means is that for example it is impossible to specify that in the concrete DSML attribute visibility should be „public“ by default.

The other frequently reported issue is inability to inherit DSML customizations. Users tend to inherit stereotypes and expect inherited stereotypes to be treated as new metamodel elements because the superstereotype has a customization class already.

Other requests that will be implemented in the nearest release include: derived tag support (ability to have derived properties just like UML derived properties); ability to set default symbol sizes in the customization class; restrict custom specification tables by element type (it should be possible to specify types of elements in the collections to display in custom specification tables).

6 Conclusions

We have discussed the recent trends in domain-specific languages and emphasized the need for a lightweight method for building domain-specific modeling languages (DSML) based on UML profiles. UML profile is preferable in many situations since it allows getting the result faster. However the UML profiling mechanism is insufficient for creating a DSML. The implemented additional customization layer for virtually transforming stereotypes into new metaclasses enables creating a full-featured DSML supporting validation of customer models.

The main benefit of our approach is that it allows defining and using a DSML in a UML tool, which leverages many standard modeling environment features like diagramming, managing model repository, analyzing model element dependencies, relationships matrices, comparing and merging models, working in a team on a single model, calculation of model metrics, support for patterns, libraries and templates, automated layout functions, model refactoring and transformations, report and code generation, plug-in platform, generating model reports, and using DSML elements in the context of UML diagrams. For achieving efficient DSML transformations, including reports and code generation, one can reuse existing approaches like *Velocity Template Language*.

However, the creators of DSML need a deep knowledge of UML metamodel for mapping domain concepts to UML concepts and OCL syntax for creating validation rules. They would also need to invest considerable amount of time into defining customizations. We understand the limitations of the current implementation and have ascertained this from the user feedback. The nearest release will include the following enhancements: functionality for creating completely new graphical symbols, setting default values for UML elements, inheriting customizations, using derived tags, and others.

7 Acknowledgements

We would like to thank Nerijus Jankevicius, who was the primary author of the idea to customize stereotypes, Tomas Juknevicius, who proposed the validation framework, and the other members of MagicDraw UML R&D team who contributed to the presented approach for creating DSMLs based on UML profiles.

References

[1] **Abouzahra, A., Bézivin, J., Fabro, M.D.D., Jouault, F.** A practical approach to bridging domain specific languages with UML profiles. *In: Proceedings of the Best Practices for Model Driven Software Development at OOPSLA'05, San Diego,* 2005.

[2] **Booch, G.** Object Oriented Design with Applications. *Benjamin Cummings, California,* 1991.

[3] **Bräuer, M., Lochmann, H.** Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. *In: 4th International Workshop on (Software) Language Engineering (ATEM'07), Nashville,* 2007.

[4] **Chen, K., Sztipanovits, J., Neema, S., Emerso, M., Abdelwahed, S.** Toward a Semantic Anchoring Infrastructure for Domain-Specific Modeling Languages. *In: The Fifth ACM International Conference on Embedded Software (EMSOFT'05), Jersey City*, pp. 35–43, 2005.

[5] **Chen, P.** The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems,* Vol. 1, No. 1, 9 – 36, 1976.

[6] **Clark, T., Evans, A., Sammut, P., Willans, J.** Applied metamodelling a foundation for language driven development. Version 0.1, http://www.vanguard-technologies.com/Services/AppliedMetamodellingV01.pdf, 2004.

[7] **Denno, P.** Model-Driven Integration Using Existing Models. *Software, IEEE Computer Society,* 20 (5), 59-63, 2003.

[8] Dresden OCL2 Toolkit : Dresden OCL2, http://dresden-ocl.sourceforge.net/.

[9] Eclipse: Eclipse Modeling Framework, http://www.eclipse.org/emf/

[10] **Esser, R., Janneck, J.W.** A framework for defining domain-specific visual languages. *In: Workshop on Domain Specific Visual Languages, in conjunction with ACM Conference on Object-Oriented Programming, Systems, Languages and Applications OOPSLA-2001, Tampa Bay,* 2001.

[11] **Evans, E.** Domain-Driven Design: Tackling Complexity in the Heart of Software. *Addison-Wesley,* 2003.

[12] **Feilkas, M.** How to represent Models, Languages and Transformations? *In: 6th OOPSLA Workshop on Domain-Specific Modeling (DSM'06), Gray, J., Tolvanen, J.-P., Sprinkle, J. (eds.), Computer Science and Information System Reports, Technical Reports, TR-37, Jyväskylä,* 2006.

[13] **Greenfield, J., Short, K., Cook, S., Kent, S., Crupi, J.** Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. *Wiley,* 2004.

[14] **Gudas S., Skersys T. Lopata A**. Approach to Enterprise Modelling for Information Systems engineering. *Informatica,* Vol. 16(2), 2005, 175–192.

[15] **Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.** Object-Oriented Software Engineering – A Use Case Driven Approach. *ACM Press, Addison-Wesley, Mass,* 1992.

[16] **Jouault, F., Bezivin, J.** KM3: a DSL for Metamodel Specification. *In: Gorrieri, R., Wehrheim, H. (eds.) Formal Methods for Open Object-Based Distributed Systems, LNCS Springer Berlin / Heidelberg,* vol. 4037, pp. 171-185, 2006.

[17] **Kapocius, K., Butleris, R.** Repository for business rules based IS requirements. *Informatica,* 2006, Vol. 17(4), 503–518.

[18] **Lagarde, F., Espinoza H., Terrier F., Gerard S.** Improving UML profile design practices by leveraging conceptual domain models. *In: 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), Atlanta,* 2007.

[19] **Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.** The Generic Modeling Environment. *In: Workshop on Intelligent Signal Processing, Budapest,* 2001.

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

[21] OMG: Meta Object Facility (MOF) Core Specification, OMG Available Specification, Version 2.0, http://www.omg.org/docs/formal/06-01-01.pdf, 2006.

[22] OMG: OMG Systems Modeling Language (OMG SysML™), V1.0 http://www.omg.org/cgi-bin/doc?formal/2007-09-01, 2007.

[23] OMG: OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF , 2007.

[24] OMG: UML Profile for MARTE, Beta 1, http://www.omg.org/cgi-bin/doc?ptc/2007-08-04 , 2007.

[25] OMG: UML Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF), Beta 1, http://www.omg.org/docs/dtc/07-08-02.pdf , 2007.

[26] **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.** Object-Oriented Modeling and Design. *Prentice Hall, New Jersey,* 1991.

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

[28] The Extensible Markup Language specification, http://www.w3.org/TR/REC-xml/

[29] The Ruby language website, http://www.ruby-lang.org/

[30] **Tolvanen, J., Pohjonen, R., Kelly, S.** Advanced Tooling for Domain-Specific Modeling: MetaEdit+. *In: Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.P. (eds.) The 7th OOPSLA Workshop on Domain-Specific Modeling, Finland,* 2007.

[31] **White, J., Schmidt, D. C., Mulligan, S.** The Generic Eclipse Modeling System. *Model-Driven Development Tool Implementer's Forum, TOOLS '07, Zurich,* 2007.

[32] **Wirfs-Brock, R., Wilkerson, B., Wiener, L.** Designing Object-Oriented Software. *Prentice Hall, New Jersey,* 1990.