Rigorously Defined Domain Modeling Languages

Emanuel S. Grant[1]*, Krish Narayanan[2], and Hassan Reza[1]

[1] Department of Computer Science
University of North Dakota
Grand Forks, North Dakota, USA
[grante, reza]@cs.und.edu
[2] Department of Computer Science
Eastern Michigan University
Ypsilanti, Michigan, USA
knarayan@emich.edu

Abstract. The use of standardized domain-specific modeling languages (DSML) in specifying software requirements and design models can lead to cost-effective development of large complex systems. DSMLs offer a vocabulary of terms and concepts that are fundamental to the problem and solution domains. The objective of this work is the development of Unified Modeling Language based DSMLs in a Model Driven Development environment. Expected benefits of using such DSML are: the reuse of reliable domain experience and artifacts; high quality models that closely represent the application domain; improved communication channels between customers and developers; reduction in the delivery times of large complex systems; improved quality and scope of code generators; and software systems that are more easily maintained.

1 Introduction

Domain-specific knowledge is often not completely represented in the models of some applications, such as engineering design [7]. The reason for this is the complexity of the knowledge and the lack of facilities in the more traditional modeling languages to represent them. The use of *domain-specific modeling language* (DSML) in specifying software systems is on the increase. This is evident from the many DSMLs being developed (e.g. [5, 8]). A reason for this shift is DSMLs offer a vocabulary of terms and concepts that are fundamental to the problem and solution domains, and the use of such DSMLs in specifying large and complex systems can be carried out in a cost-effective manner [5].

The availability of: (1) model driven development framework (*Model Driven Architecture* [6] (MDA)); (2) standardized modeling notation (*Unified Modeling Language* [4] (UML)); (3) mechanisms to specialize such modeling notation, (UML *extension mechanism* (EM)); (4) artifacts from domain analysis [3]); and (5) formal/informal notation integration techniques ([2]), make it possible to define graphical-based DSML that have well-defined syntax and semantics.

The MDA is a newly defined model-centric framework from the Object Management Group[3]. At the heart of MDA principles is the definition of: *computational independent models* (CIMs), which offer views of the system from the users' perspectives; *platform independent models* (PIMs), which offer views of the system independent of any technological platform that the system may be implemented on; *platform specific*

* This work was partially funded under the North Dakota NASA-EPSCoR Opportunities Program.

[3] *www.omg.org*

models (PSMs), which offer views of the system from the intended implementation technological platforms; and *mappings* between the various models. These models may be expressed in the UML notation.

The UML is a set of graphical and textual notations for modeling various views of software systems, using object-oriented (OO) concepts. The UML specification offers a set of syntactically well-defined modeling notations, and is a general purpose modeling language. The UML's EMs are used to tailor the standard UML concepts for specific enterprise, domain or application requirements. The UML EMs were specifically defined to meet the requirements of some projects which require features beyond those defined in the UML specification.

It has been proposed that DSML developers take an approach that is based on the MDA. In this approach DSML developers will create PIMs of the domain requirement CIMs, using the UML notation. Constraints will be applied to the models (PIMs and CIMs) in order to formally determine the CIM to PIM *transformation mapping*. The analysis level PIMs are transformed to design level PIMs by *refinement mappings*. The PIMs are then transformed to the desired PSMs in preparation for automatic transformation to the intended code. The mappings between the models may be verified for correctness by analysis of the integrated formal notation constraints.

The remainder of this paper is as follows: Section 2 outlines the framework for DSML development; Section 3 presents the components of DSML, its syntax and semantics; Section 4 defines the process for constructing and using DSMLs; Section 5 looks at related works; and Section 6 concludes with a look at some benefits of using DSML.

2 Framework for DSML development

In this work DSMLs are represented as UML *profiles* of PIMs, PSMs and the mappings between them. The goal is to use features of the UML, namely meta-model elements, extension mechanisms, and profiles within the framework of MDA to define DSMLs as a tool for the application engineer. The components of such DSMLs are to be used at the requirement analysis and design stages of software development, within a tool environment (e.g. Rational Rose® [1]). DSMLs provide domain-specific modeling constructs (PIMs and PSMs) to create application models, which may be analyzed before being translated to code. In such an environment, application engineers will develop models using constructs that directly reflect domain concepts, and these models will incorporate expert experiences related to development decisions.

A high-level architectural description of the development environment for DSML, which is termed *Rigorous Domain-Specific Software Engineering* (RDSSE) in this work, is illustrated in Figure 1. A DSML, in RDSSE, is a packaged set of UML model elements and domain meta-models that have been precisely developed for use in a specific problem domain. The semantics and syntax of DSML are specified with informal and formal notations — the informal UML meta-models, and formal UML Object Constraint Language (OCL) [4] constraints. The *feedback* arrows between the activities in RDSSE are to facilitate modifications to the DSML that may arise from errors, inconsistencies, omissions, and new requirements that have been uncovered during execution of the RDSSE activities (*Domain Analysis and Design, Domain Language Engineering* and *Application Engineering*). The feedback activities provide support for backward compatibilities when changes occurred in the requirements of the domain.

3 DSML Components

DSMLs are presented as a set of components — domain meta-models (descriptions of PIMs, PSMs) and domain rules (adaptation, composition and transformation), which defines the syntax and semantics of

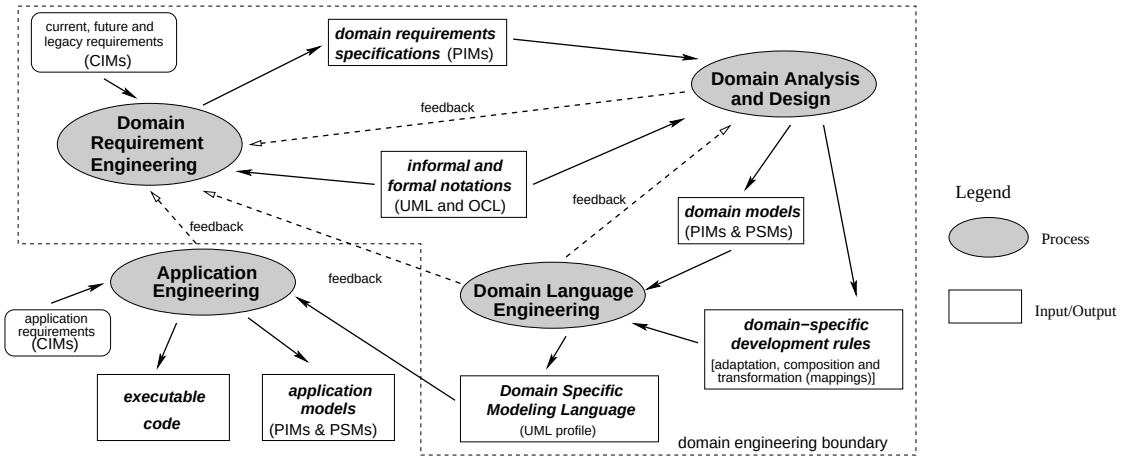

Fig. 1. Rigorous Domain Specific Software Engineering (RDSSE)

DSML. The relationships of DSML components are presented in Figure 2 as a UML profile (*DSML*) of profiles (*Stereotypes, Class Diagram meta-model*, etc.). The meta-models of the domain PIMs and PSMs are created during the language engineering activity (see Figure 1).

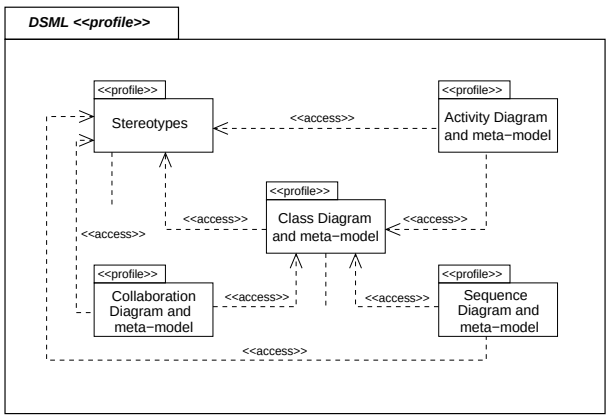

Fig. 2. DSML components

3.1 Domain models

Developers will select a subset of the UML models (class diagram, use case diagram, activity diagram, sequence diagram, collaboration diagram, state chart diagram) for development within the specified do-

main. Creation of domain models (PIMs and PSMs) is accomplished during the *Domain Analysis and Design* activity of Figure 1. For each type of model in the selected set of UML models; the commonalities (across all application models in the domain) are identified as *patterns* of the domain. These patterns of commonalities constitute the mandatory structure of the particular model for the domain. The variable components of the model are also included in the models, with rules (constraints) that specify how they should be structured with the common (*pattern*) components.

3.2 The domain rules

During software development, developers will implicitly and explicitly apply *rules* in creating the models of the application. Some of these rules are from the requirements and others are from the experience gained from work on previous software development projects. In the *Domain Language Engineering* activity of Figure 1 the *Domain-Specific Development Rules* are used to define and constrain the syntax and semantics of the DSML, and specify the mappings for PIM to PIM and PIM to PSM transformations. The rules are applied to the concepts (static and dynamic) of the domain. Static concepts are the objects and associations, and dynamic concepts refer to system services of the domain. The rules are expressed as OCL statements (when possible, and as precise textual statements when not possible) in the stereotyped UML model elements.

In RDSSE *Domain Language Engineering* three types of rules are specified:

1. *Adaptation Rules* - determine which UML model element a domain concept is to be mapped to, and the inherited syntax and semantics constraints.
2. *Composition Rules* - determine which domain concepts are mandatory and optional in the application models of the domain. Execution of the composition rules leads to the definition of PIMs and PSMs that capture *patterns* in the domain. These rules result in the definition of composite domain concepts from elementary domain concepts.
3. *Transformation Rules* - determine how the PIM to PIM, PIM to PSM and PSM to PSM mappings are conducted. The PIM to PIM and PSM to PSM transformation rules are of two types; *model refinement* and *model association* rules.

The domain rules also specify how models may be instantiated from the DSML meta-models (e.g. an instantiation of an application class diagram from the class diagram meta-model, or instantiation of an application model element from a stereotyped model element).

3.3 Stereotypes

The *Stereotypes* profile of Figure 2 contains the domain-specific semantics as defined by the *Domain-Specific Development Rules*. The semantics is in the form of stereotype OCL constraints, denotational mappings, and precise textual statements. The stereotypes specify how UML meta-model elements are transformed into domain-specific meta-model elements by the application of the EMs (*constraint, stereotype*, and *tag definition*) in defining the semantics of the DSML. This is a PIM to PIM transformation.

An example of a stereotype for the *item* class of the *Checkin-Checkout* domain is presented in Table 1. The stereotype was developed for an example domain (*Checkin-Checkout* [3]). This domain is a family of applications that is characterized by a set of users who will be associated with the functions of checking in and checking out items. An example application in this domain is a car rental application that facilitates a customer (*user*) renting (*checking out*) a motor vehicle (*item*) and returning (*checking in*) a motor vehicle.

The meta-attributes of the stereotyped model elements are explicitly assigned values that aid in the specialization of the model element semantic. he semantics of the stereotype (≪*item*≫) is defined by the inherited semantics of its: *Base Class*, and the semantics of the owned operations (*request_checkin, update_item, create_item* and *verify_item*). Stereotypes for the operations are also defined. The textual stereotypes may be represented in a graphical format. The graphical format present the stereotypes as *realized* specializations of the UML model elements' base classes.

Table 1. Class *Item* stereotype

Stereotype	item
Base Class	Class
Parent	N/A
Tag	mandatory
Constraint	**context** ≪item≫ **inv:** self.isAbstract = false **and** self.isLeaf = true **and** self.isRoot = true **and** self.mandatory = true **and** self→isUnique(self.item_code) **and** self.≪user≫→size() = 1) **and** self.≪description≫→size() = 1 **and** self.≪transaction≫→size() ≥ 0) **and** self.≪item_category≫→size() > 0 **and** self.≪item_code≫.multiplicity = 1 **and** self.≪item_code≫.changeability = frozen **and** self.≪item_code≫.visibility = public **and** self.operation→includesAll(request_checkin, update_item, create_item, verify_item)

3.4 Domain meta-models

The meta-models of RDSSE are specifically developed for the language engineering process, and are not intended to be explicitly used in *Application Engineering*. The meta-models are used by tool developers in interpreting the syntax of the DSML. In RDSSE meta-models are created with the defined domain stereotypes and UML model elements of the domain models. Each meta-model is a graphical description of the respective domain models, i.e. the meta-model represents the structural description of the domain model.

The meta-models of DSMLs (see Figure 2) are centered around the class diagram meta-model, as a complete domain class diagram captures information from multiple views of a software system. A *complete* domain CD is one in which all relevant classes, attributes, operation signatures, and associations are included. The *un-ended* lines of Figure 2 indicate there may be other sub-packages and relationships, (i.e. additional meta-model profiles) that are not shown in this particular description.

4 Engineering DSML

In order for the development and use of DSMLs to be successful there has to be a well defined process for creating DSMLs. The process for engineering a DSML in this work is made up of six key tasks. This process, is illustrated in Figure 3 (reproduced from [3]) in the form of an UML activity diagram. The six tasks associated with the development of DSML in the RDSSE are: (1) *Create SC* (Static Concept) *Stereotype*, (2) *Create DC* (Dynamic Concept) *Stereotype*, (3) *Package Domain Stereotype*, (4) *Create Domain Meta-Models*, (5) *Package Domain Meta-Models*, and (6) *Package DSML*.

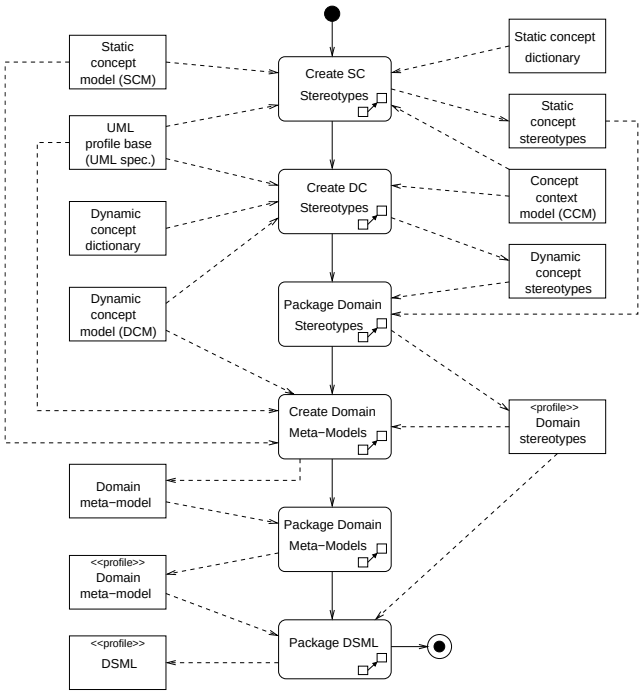

Fig. 3. DSML development process

The first two task of DSML development (*Create SC Stereotype* and *Create DC Stereotype*) are initiated in the domain analysis activity of RDSSE. These two tasks are presented in the *Language Engineering* activity of RDSSE because of the importance of the stereotype definition activity to DSML development, and only the initial aspects of the stereotype definition are carried out in the domain analysis activity. Packaging of the stereotypes involves the definition of a *profile* for the three sets of stereotypes (static concepts, use cases, and operations). The sets of stereotypes are individually packaged in profiles and these profiles are in turn packaged as a profile of profiles (*Domain Stereotypes*).

Creation of the meta-models of the PIMs and PSMs is an activity that is described as *Define Graphical Icon for Stereotype*, and *Establish Associations Between Icons*. The domain meta-models are intended to provide a graphical representation of a sub-set of the information contained in the domain stereotypes.

Packaging of the meta-models is carried out in a similar manner to that of the domain stereotypes. Finally, packaging of the DSML is obtained by extending the meta-model profile package to include the DSML stereotype profile package. A detailed description of the processes for developing DSML in the RDSSE environment is presented in [3].

4.1 Using DSML

DSMLs, as presented in this work, are intended to be used by application engineers in a CASE tool environment. Tool developers would encode the DSML (syntax, semantics) in the CASE tool in a manner that is identical to that which is currently done with the UML specification, with one major exception. This exception is with respect to the domain pattern meta-models that constitute the syntax of the DSML. The meta-models and stereotypes capture development knowledge not present in the general purpose UML meta-model.

The semantics of a DSML is presented to a tool developer in the format of a set of stereotypes that are constrained by OCL expressions and denotational mappings. This is precise and is directed toward a single interpretation for each DSML element. Consequently, the encoding of the semantics of a DSML in all CASE tools will result in all such tools expressing the semantics of the DSML in an identical manner.

The stereotyped and UML generic icons for the DSML would be listed on the drawing palette of the DSML tool. The mandatory concepts may be presented in a different color or text font from that of the optional ones. The application developer would select the desired icons from the palette and place them in the workspace area. The tool would then automatically include the mandatory concepts associated with the selected icons. This process would continue until the developer has created models that satisfy the requirements for the application.

5 Related works

Sprinkle and Karsai [8] address the issue of upward compatibility of domain models, i.e., the preservation of semantics of old models in an evolved domain-specific language. The usage of domain-specific languages, in this case, visual languages, is very promising because of its incorporation of some fundamental domain concepts. But, this in itself can prove disadvantageous if the domain evolves, even worse if it evolves frequently. Old domain models might not adhere to the new/modified syntax/semantics of the evolved language. The paper introduces a meta-modeling language to aid in the domain evolution process, serving more as an interface and less as an automated transformation tool. It aids a meta-modeler in deriving algorithms for mappings of syntactical patterns of a language, which are described in the meta-model. These mappings are used to draw up a sequence of transforms to map the old domain models to their new counterparts. The language has strong roots in graph transformation and graph rewriting mechanisms.

Sprinkle and Karsai work is an important advancement step from our perspective. The domain evolution concept discussed correlates with the feedback arrows in the RDSSE process from the domain language engineering activity. Changes in the language should reflect in the new models developed and the old models that are reused. Though we have not defined how and what is involved in the feedback activities of RDSSE it is acknowledged that the ability to respond to language evolution is fundamental ti its successful and long term usage in software development. The work of Sprinkle and Karsai is of great relevance to our definition and use of DSML.

Dae-Kyoo, France and Ghosh [5] proposed the development of UML-based languages for specifying domain-specific patterns, which they term *Role-Based Meta-modeling Language* (RBML). In their work

a RBML is a specification *"that defines a sub-language for a particular type of UML diagram"*. Similar to RDSSE DSMLs, RBMLs are intended to be languages for families of applications within specified domains. But unlike our DSML, RBMLs do not explicitly specify which UML model elements are to be used in application models, but specify a set of properties that *must* be satisfied by a component in order for it to play the *role* associated with the set of properties.

While Dae-Kyoo *et al.* work with RBML seems to be congruent in many aspects with our DSML they differ in two major ways. RBMLs seem to require users (application developers) be able to explicitly associate intended application components with properties in the domain patterns. Our DSML require no such skill on the part of the user, in fact our DSML can be immediately used by any developer who already has knowledge in the use of the UML. Secondly, Dae-Kyoo *et al.* introduces *new* concepts (new to the UML), which are used to supplement the UML semantics in their UML role model patterns. Our DSMLs rely solely on the UML model elements and derive *new* domain-specific concepts by the application of the UML EMs on the desired model elements.

6 Conclusion

In this report the components and process for developing domain-specific modeling languages (DSML) that are based on the MDA framework and UML specifications are presented. The expected benefits of using DSML are: a graphical modeling language for application development that reduces the time to deliver a complete system; the capture of requirement analysis and design decisions and extensive reuse of domain artifacts; application models that have been rigorously analyzed; and software applications that are more easily maintained. In addition, the DSML are intended to be extensible and easily maintained,

The work presented in this report on developing UML-based DSML within the MDA framework has been implemented on a text book type problem domain, the *Checkin-Checkout* domain and a research problem domain, NASA's code generator for state space estimators (detailed in [3]). Currently, work is being proposed for developing a RDSSE DSML for a real world problem domain (NASA's altitude control systems).

References

1. Michael Boggs and Wendy Boggs. *Mastering UML with Rational Rose 2002*, volume 0782140173. Sybex, 2002.
2. Robert B. France, Jean-Michel Bruel, Maria M. Larrondo-Petrie, and Emanuel Grant. Rigorous object-oriented modeling: Integrating formal and informal notations. In *Proceedings of the 6th International AMAST Conference*, 1997.
3. Emanuel S. Grant. *Defining Domain-Specific Object-Oriented Modeling Languages as UML Profiles*. PhD thesis, Colorado State University, Ft. Collins, Colorado, USA, December 2002.
4. Object Management Group. *OMG Unified Modeling Language (UML) Specification*. Object management Group, Needham, Massachusetts, USA, uml 1.5 edition, March 2003.
5. Dae-Kyoo Kin, Robert France, and Sudipto Ghosh. A UML-based language for specifying domain-specific patterns. *Journal of Visual Languages and Computing*, 15:265–289, Jan. 2004.
6. Joaquin Miller and Jishnu Mukerji (ed.). Model driven architecture (MDA). Technical Report ormsc/2001-07-01, Architecture Board ORMSC, (OMG), July 2001.
7. K. Narayanan, F. Mili, and D. VandenBossche. Domain knowledge in engineering design: Nature, representation and use. In R. Roy, editor, *Industrial Knowledge Management: A Micro-Level Approach*, pages 55–72. Springer Verlag, 2000.
8. Jonathan Sprinkle and Gabor Karsai. A domain-specific visual language for domain model evolution. *Journal of Visual Languages and Computing*, 15:291–307, Jan. 2004.