A Meta-Model Approach to Defining UML-Based Domain-Specific Modeling Language

Emanuel S. Grant

Abstract— **The use of general purpose modeling languages (GPMLs) in specifying software applications has given way to the use of domain-specific modeling languages (DSMLs). DSMLs offer a vocabulary of terms and concepts that are fundamental to the problem and solution domains, whereas GPMLs constructs are usually too generic to be directly applied in some domains. Many DSMLs are high-level textual programming languages, which offered little support for modeling at the analysis, and design phases of application development. The objective of this work is to develop semi-formal graphical DSMLs, which are to be used at the analysis and design stages of application development. The benefits derived from such DSML are reuse of domain artifacts; reduction in delivering completed products; rigorous analysis of domain applications; and more maintainable applications.**

Index Terms— **Domain modeling language, meta-model, domain analysis.**

I. Introduction

THE use of general purpose modeling language (GPMLs) in specifying software applications has given way to the use of domain-specific modeling language (DSMLs). This is evident from the many DSMLs being developed and used (examples are [1, 2, 3]). DSMLs offer a vocabulary of terms and concepts that are fundamental to the problem and solution domains, whereas GPMLs constructs are usually too generic to be directly applied in a solution for some problem domains. Review of some DSML show that they: (1) are based on textual notations; (2) lack a clear definition of the underlying syntax and semantics; and (3) are designed for use at the implementation phase of software development.

These observations demonstrate the limited use of such DSML. In domains where graphical notations are extensively used, textual notations add ambiguities. In others, analysis and design models are required as key products of the software development process. While in some, rigorous analysis of application models is required.

The availability of standardized modeling notations for object-oriented (OO) development (the Unified Modeling Language (UML) [4]) and a mechanisms to tailor such modeling notation, (the UML extension mechanism (EM)), make it possible to define graphical-based domain-specific modeling language (DSML) that have a well-defined syntax and semi-formal semantics. Such DSML are made more practical by having artifacts from domain analysis (DA) (e.g. commonality analysis [5]), and the capability to integrate formal specification techniques (FST) [6] with the informal modeling notations.

The UML [4] is a set of graphical and textual notations for modeling various views of software systems, using OO concepts. The UML's EM is used to tailor the standard UML concepts to specific requirements. Such mechanisms offer support, for the UML to be structured around a core set of constructs. This core UML will then form the base for the derivation of other UML concepts and constructs [7].

The application of a coherent set of UML EMs that is driven by the requirement of specific application domains results in a UML profile. A UML profile is a package that contains stereotyped (and non-stereotyped) model elements that have been assembled to satisfy a set of modeling requirements for a particular domain.

The definition of semi-formal graphical modeling language components that are specific to particular domains is the goal of this work. In this report, DSML will be represented as UML profiles. The intention is to use the features of the UML: meta-model elements, EMs, and profiles to define DSMLs as a tool for application engineers.

DSMLs provide domain-specific modeling constructs to create application models that can be analyzed before being translated to code. In such environments, 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 such a development environment, which is termed Rigorous Domain-Specific Software Engineering (RDSSE) in this work, is illustrated in Figure 1. Only the Domain Language Engineering activity of Figure 1 is considered in this report.

This paper is organized as follows: Section 2 introduces and defines the concept of domain-specific modeling languages, as presented in this report; Section 3 illustrates the components that constitute a DSML on an example application domain; and Section 4 concludes with a look at some related works and the benefits of using DSML.

II. DOMAIN-SPECIFIC MODELING LANGUAGE

There is a large amount of work on DSML (see [1, 2, 3]), which holds promise for improving the quality of software products. Notwithstanding the successful application of DSML technologies, it has been realized that many DSMLs are chiefly high level programming languages. Work with UML profile has added another dimension to the application

Manuscript received December 08, 2011; revised January 31, 2012.
Emanuel S. Grant is an Associate Professor with the Department of Computer Science, University of North Dakota, Grand Forks, North Dakota, USA. Phone 701.777.4133, Fax 701.777.3330, grante@aero.und.edu.

of domain-specific technology.

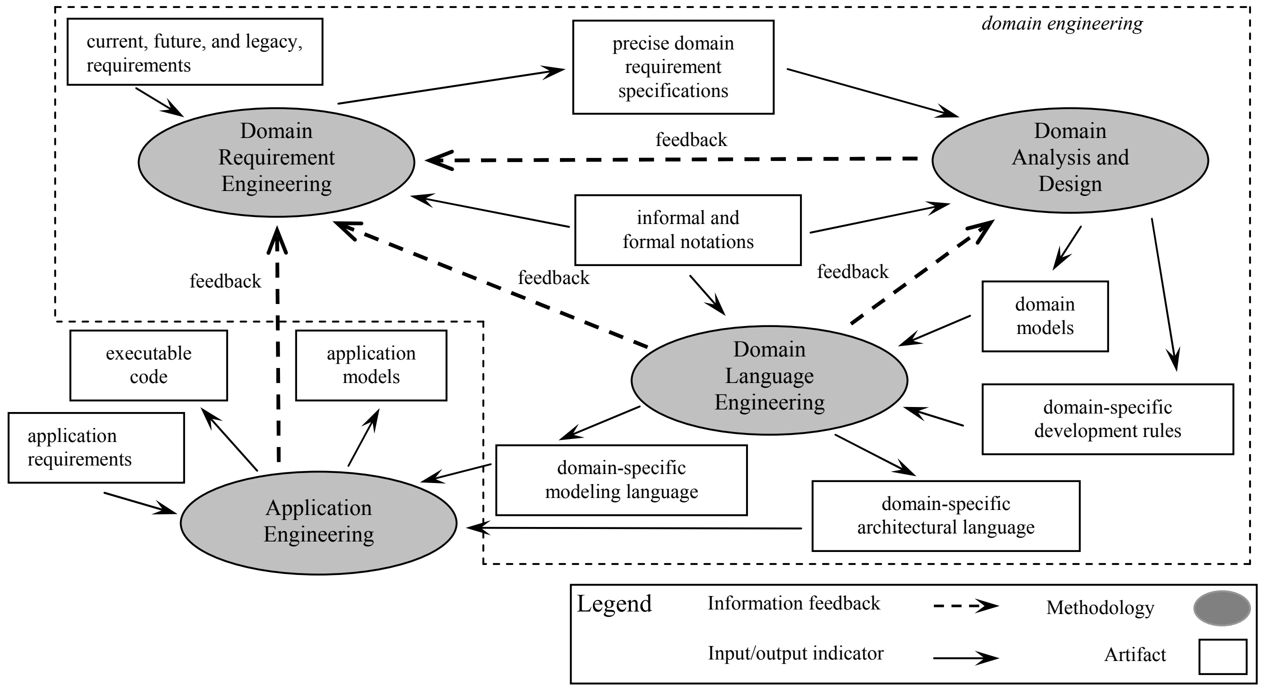
domain engineering
current, future, and legacy, requirements
precise domain requirement specifications
Domain Requirement Engineering
feedback
Domain Analysis and Design
informal and formal notations
feedback
feedback
feedback
domain models
executable code
application models
Domain Language Engineering
domain-specific development rules
application requirements
domain-specific modeling language
Application Engineering
domain-specific architectural language
Legend
Information feedback
Methodology
Input/output indicator
Artifact

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

A. Requirements for DSML

In order for DSML technology to be exploited in any domain, there are a number of requirements that must be met for the technology to be successfully applied.

The first requirement is that the domain of concern must be one that is mature, well defined, and is comprised of a useful set of current and required applications [8]. This requirement is based on the necessity that the DSML must have some long-term intended use.

A second requirement is that the DSML must encompass all known and relevant terms of the domain. Such terms are the names of all domain-specific concepts, objects, relationships, operations, functions, etc. that may appear in any domain application specification at the analysis, and design phases of software development. This requirement is based on the necessity for the vocabulary, syntax, and semantics of DSL terms to remain as stable as possible.

A third requirement for DSML is that it provides abstraction for the domain concepts at the analysis and design stages. Domain abstraction is chiefly concerned with the mapping of various external views of the domain applications to a set of functional representations [3].

A fourth requirement for DSML is that it should be evolvable. With the passage of time, there will be new requirements or the domain may be widened to include features and concepts that were not a part of the initial set of requirements, features, and concepts. The DSML for such domains must be extended to accommodate these new requirements, features and concepts.

B. DSML Syntax

The syntax for DSMLs is obtained from the UML syntactic base. The syntactic base of a DSML is defined by: (1) a syntactic domain, (2) a mapping from the syntactic domain to a set of domain constructs (icons, picture, etc.), and (3) a mapping from the domain constructs to the concepts of the domain. The syntactic domain defines the visual representation for the language, i.e. the language notation.

The UML's syntactic domain is a sub-set of constructs from the 2-dimensional space of geometric figures, and alphanumeric characters. The sub-set of figures include, lines, points, and arrows that are used to represent the model elements of the UML abstract syntax, and models that are derived from the abstract syntax. The notation of a visual language is the set of iconic elements (concrete syntax) that are mapped to concepts of the language. Figure 3 gives examples of the concept to concrete syntax mapping for the UML.

The syntactic domain for a DSML may be identical to the UML's, a sub-set of the UML's, or taken from a different domain. Similarly, the iconic elements used to define the abstract syntax of the DSML may be identical to the UML's, a sub-set of the UML's, or from a different set of icons. The use of a syntactic domain, and syntactic elements that are different from the UML's are permitted in UML. This is achieved via the EM attribute icon that is of the type Geometry and is defined as a ". . . geometric description for an icon to be used to present an image of a model element branded by the stereotype." [4].

C. DSML Semantics

A semantic domain for the UML is not explicitly given in the UML specification [4], but this can be deduced, for each model element, from the semantic definition. The semantic definition for a Class states that a "class is a description of a set of objects…" [4] and an association is defined as "a semantic relationship between classifiers." [5], where the "instances of an association are a set of tuples relating instances of the classifiers" [and each] "tuple value may appear at most once" [5]. These semantic definitions imply that UML class may be semantically mapped to the mathematical concept of sets, and association to the mathematical concept of relationship [9].

The approach taken in France et al. [9] and adapted in this work, is to map the domain concept to a mathematical concept, with similarities. Examples of this mapping from a domain (the UML) concept to a well-known and understood domain (mathematical) concept is illustrated in Figure 3, where the UML Concept is mapped to a Semantic Domain concept that is mathematical.

In defining the semantics of DSML a mapping from the DSML concept to a formal (mathematical) domain concept that exhibit similar properties to that of the domain concept will be attempted. This is the approach outlined in the following works [10, 11]. In the case where it is not possible, to map a domain concept to a mathematical concept the semantics of the domain concept will be presented in the form of concise textual descriptions. This approach explains the classification of the DSMLs of this work as semi−formal. The semantics of the DSML is expressed a mixture of formal statements and concise textual statements.

III. DSML COMPONENTS

A DSML is presented as a set of components: domain meta-models and domain rules that constitute the syntax and semantics of the DSML. The structure for DSML is represented as a profile (DSML) of packaged profiles (DSML Stereotypes, Class Diagram meta-models, etc.). The DSML profile package contains meta-models of the domain models that are created during the domain analysis and design activity (illustrated in Figure 1). The meta-models are centered round the class diagram (CD) meta-model, as a complete domain CD contains information from multiple views of a software system. A complete domain CD is one in which all classes, attributes, operation signatures, and associations that are relevant to the domain problem under consideration are included. The un-ended lines of Figure 4 indicate there may be other sub-packages and relationships, (i.e. additional meta-model profiles) that are not shown in this particular description.

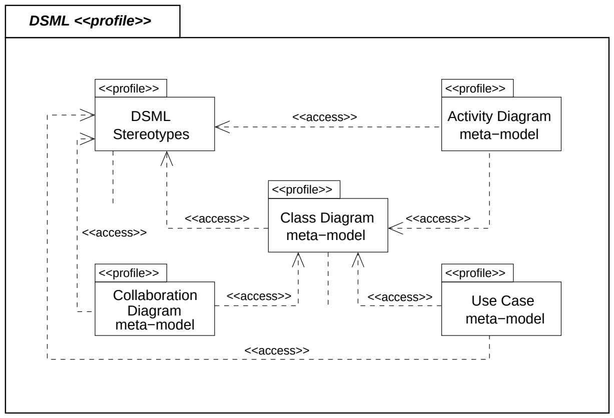

Fig. 2. DSML Architecture

A. Domain Meta-models

In RDSSE, a meta-model is created for each of the domain UML models developed. These meta-models are specifically developed for the Language Engineering process, and are not intended to be explicitly used in Application Engineering. The meta-models are to be used by tool and language developers. Language developers will use the meta-models in defining the syntax of the language. Tool developers will use the meta-models in interpreting the syntax of the DSML, and facilitate automatic generation of application model components within the tool environment.

The meta-models are composed of graphical representations of the domain stereotypes, the associations between the stereotypes, and the multiplicities on the associations. The components of the meta-models must fully comply with the syntactic and semantic definitions of the UML, for the respective models and model elements used. The multiplicities in a meta-model stipulate the number of model elements that may appear in an associated application model. This is because the instances of the meta-model elements (stereotypes) are the stereotype-base class model elements. This is consistent with the principle that the instances of the meta-model (instances of the stereotype) resides at the model level (MOF2 level M1) and the meta-model elements (stereotypes) reside at the meta-model level (MOF level M2) [4]. The UML Meta-Object Facility meta-model hierarchy is represented in Table 1.

TABLE I UML META-MODEL

Meta-model level	Example
M3 meta meta-model	MOF meta-class (description of the meta-model)
M2 meta-model	UML meta-model class (description of the model)
M1 model	UML attribute (description of the instance)
M0 instance	<rental_cost, 4.50>

B. Meta-model Example

Examples of a CD and its corresponding meta-model are illustrated in Figure 3 and Figure 4 respectively. The CD of Figure 3 was developed from the analysis of an example domain, namely *Checkin-Checkout* [12]. This domain relates to a family of applications that are characterized by the following features:

- An administrator of the system (e.g. librarian in a library system);
- A set of users who will be associated with the functions of checking in and checking out (e.g. customer in a car rental system);
- A set of items, with unique identifiers, that will be checked in and/or checked out (e.g. videos in a video rental system);
- A description of the items (e.g. book title information in a library system);
- A set of policies associated with the administrator, items, users, and transactions (e.g. age limit of customers in a car rental system);
- A set of transaction information (e.g. record of a rental receipt in the video rental system);
- A set of categories for users, items, and transactions (e.g. fiction or non-fiction categories);

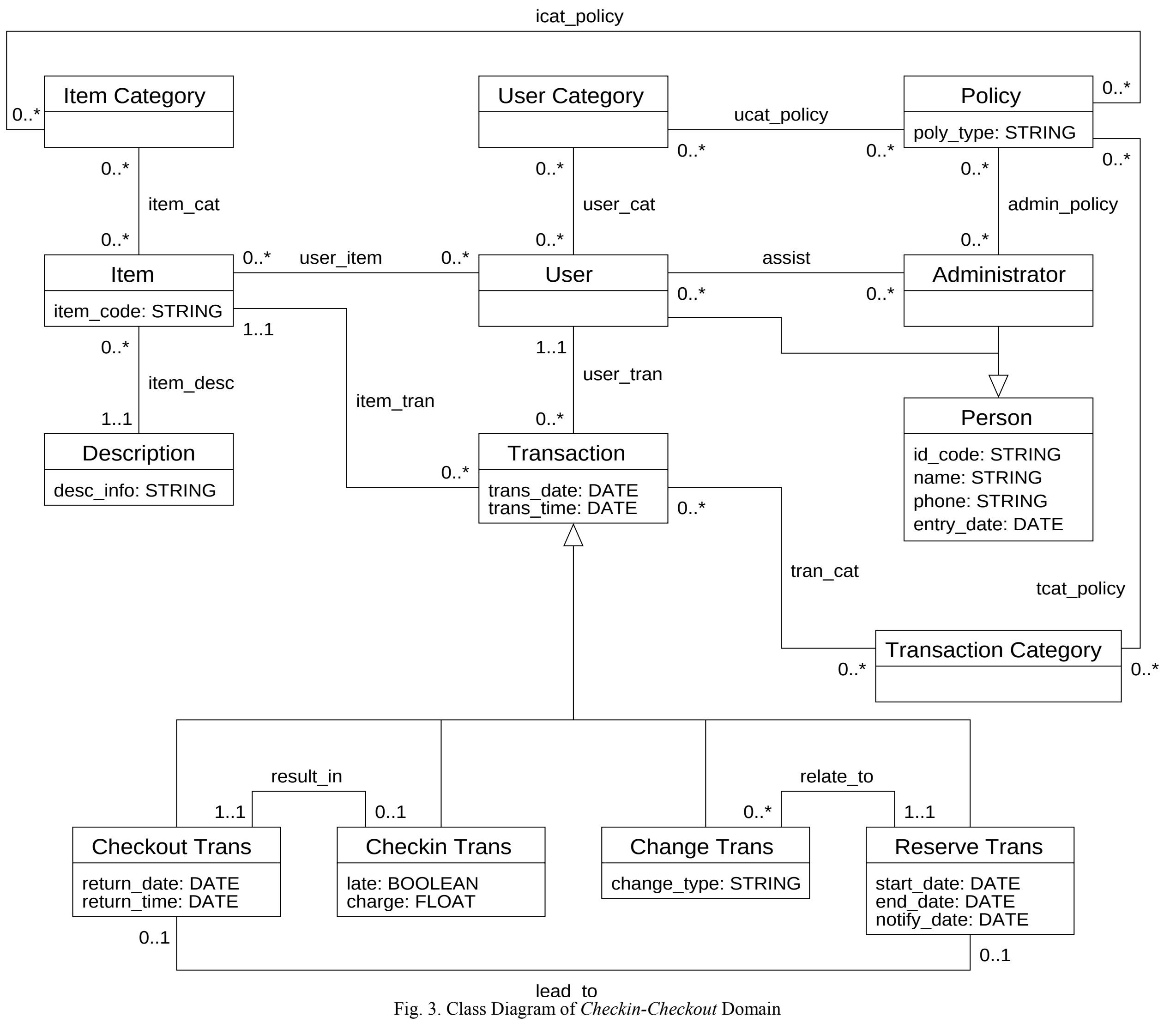

Fig. 3. Class Diagram of *Checkin-Checkout* Domain

A. The Domain Rules

During software development, the developers usually (implicitly and explicitly) apply rules in defining the models of the application. Some of these rules are from the requirements and others are from the experience gained during 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. The rules are applied to the concepts (static and dynamic) of the domain. The rules are expressed as OCL constraints (when possible) in stereotypes of UML model elements. Within RDSSE Domain Language Engineering, three types of rules are specified:

Adaptation Rule - determines which UML model element a domain concept is to be mapped to, and how the inherited syntax and semantics are to be constrained. Adaptation rules also specify what denotation mappings [13] are to be applied to the concepts to refine its semantics.

Composition Rule - determines which domain concepts must be included in an application model of the domain.

Execution of the composition rules leads to the definition of models that capture patterns of the domain. These rules result in the definition of composite domain concepts from elementary domain concepts (UML model element).

Refinement Rule - determines which domain model elements may be optionally included in domain application models. The refinement rules are applied after the composite rules have been executed. The refinement rules result in further definition of composite domain concepts from elementary and composed domain concepts. The refinement rules facilitate the identification of optional UML model elements and variations in the domain patterns.

The DSML Stereotypes sub-package of Figure 2 contains the domain-specific semantics as defined by the Domain-Specific Development Rule of Figure 1. The semantics is in the form of stereotype constraints, denotation mappings, and concise textual statements. These constraints also specify how models may be instantiated from the DSML meta-models and meta-model elements (e.g. an instantiation of an application CD from the DSML CD meta-model, or an instantiation of an application model element from a stereotyped model element), by identifying the mandatory (common) elements for the application models. The stereotypes demonstrate how the UML meta-model elements are transformed into domain-specific elements by application of the EMs (constraint, stereotype, and tag definition) to define the semantics of the DSML.

B. Stereotype Example

An example of a stereotype for the item class of the example domain *Checkin − Checkout* is presented in Table II. The meta-attributes of the stereotyped model elements are explicitly assigned values that aid in the specialization of the model element semantic definition. The meta-attribute mandatory of the stereotype ≪item≫ has been introduced as a tag definition, which is now available to be applied to any defined stereotype of the DSML. The tag definition is illustrated in Table III.

The semantics of the stereotype (≪*item*≫) is defined by the inherited semantics of its (1) Base Class, (2) uniqueness (*self* → *isUnique*(*self.item_code*)), and (3) the semantics of the associated operations (*request_checkin*, u*pdate_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 presents the stereotypes as realized specializations of the UML model element base classes.

TABLE II META-CLASS STEREOTYPE

Stereotype	*item*
Base Class	Class
Parent	N/A
Tag	*mandatory*
Constraint	context ≪*item*≫ inv: self.is*Abstract* = false and self.*isLeaf* = true and self.*isRoo*t = 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*)

The DSML syntactic (meta-models) and semantic (stereotypes) basis are presented as the foundation for DSML definition, with the inputs to the definition process for DSML being the domain models and rules. The output of the domain language engineering process is a set of meta-models that defines the syntax of the DSML, and the stereotypes that define the semantics.

TABLE III: TAG DEFINITION MANDATORY

Tag	*mandatory*
Stereotype	*item, user, description*, etc.
Type	UML::Datatype::Boolean
Multiplicity	1

C. Related Works

Desmond D'Souza et al. [14] argued that the introduction of the profile concept in the UML was redundant, as the specified mechanism for profiles could be achieved by using the UML concepts of (1) package, and package-import/generalization, (2) frameworks, and (3) OCL. This assertion was debatable, but it is a moot point today as the profile concept has been incorporated into the UML specification. The current profile concept is not a new concept, as it is defined from existing UML concepts of package and stereotype. The UML list a set of predefined package stereotypes that includes ≪*profile*≫. This approach incorporates some aspect of D'Souza's et al. view in [14], and the original profile definition in the UML.

The approach taken by Steve Cook et al. in Defining UML Family Members Using Prefaces [15] is a very high-level description of their interpretation, of what, the UML family of languages should contain and its semantics. They conceded that:

". . . the intent of the OMG's activity on profiles is very similar to the intent [of prefaces] and over time it may prove appropriate to unify the concepts profile and preface" [15].

The key aspects of their definition of a preface are the intended contents, which include:

- meta-level definitions of concepts, relationships between concepts, features of concepts, and well-formedness rules of the concepts,
- specializations and extensions of meta-models, abstract syntax, semantics, and allowed transformations and generations of the preceding, and the semantics of the models, which they suggest should be defined in the form of inference and axiomatic rules.

It is expected that prefaces will be very large because of the wide range of subjects that are contained, thus Cook et al. propose that prefaces will be organized into packages. This idea is also fundamental in D'Souza et al. [14], and is exploited in the work of this report.

IV. CONCLUSION

In this report, the UML profile is used to define semiformal domain-specific modeling language express the syntax and semantics of the DSML. The benefits derived from using DSML profiles are:

- They are graphical modeling languages used by application developers that reduce the time to deliver a complete system;
- They are extensible and easily maintained,
- They captures analysis and design decisions and facilitates extensive reuse of domain artifacts; and
- The models can be rigorously analyzed.

The UML profile was selected as the format for the DSML because the UML has evolved into the de facto modeling notation for object-oriented system development, and the profile concept requirements, as set out in the OMG's Requirements for Profiles are congruent with the requirements for a DSML, as defined in this work. The DSML profile extensibility is achieved by the separate packaging of the domain-specific semantics and abstract syntax meta-models, as illustrated in Figure 2. Some decisions a domain expert makes in developing models within the domain are captured in the meta-models and semantics of the DSML.

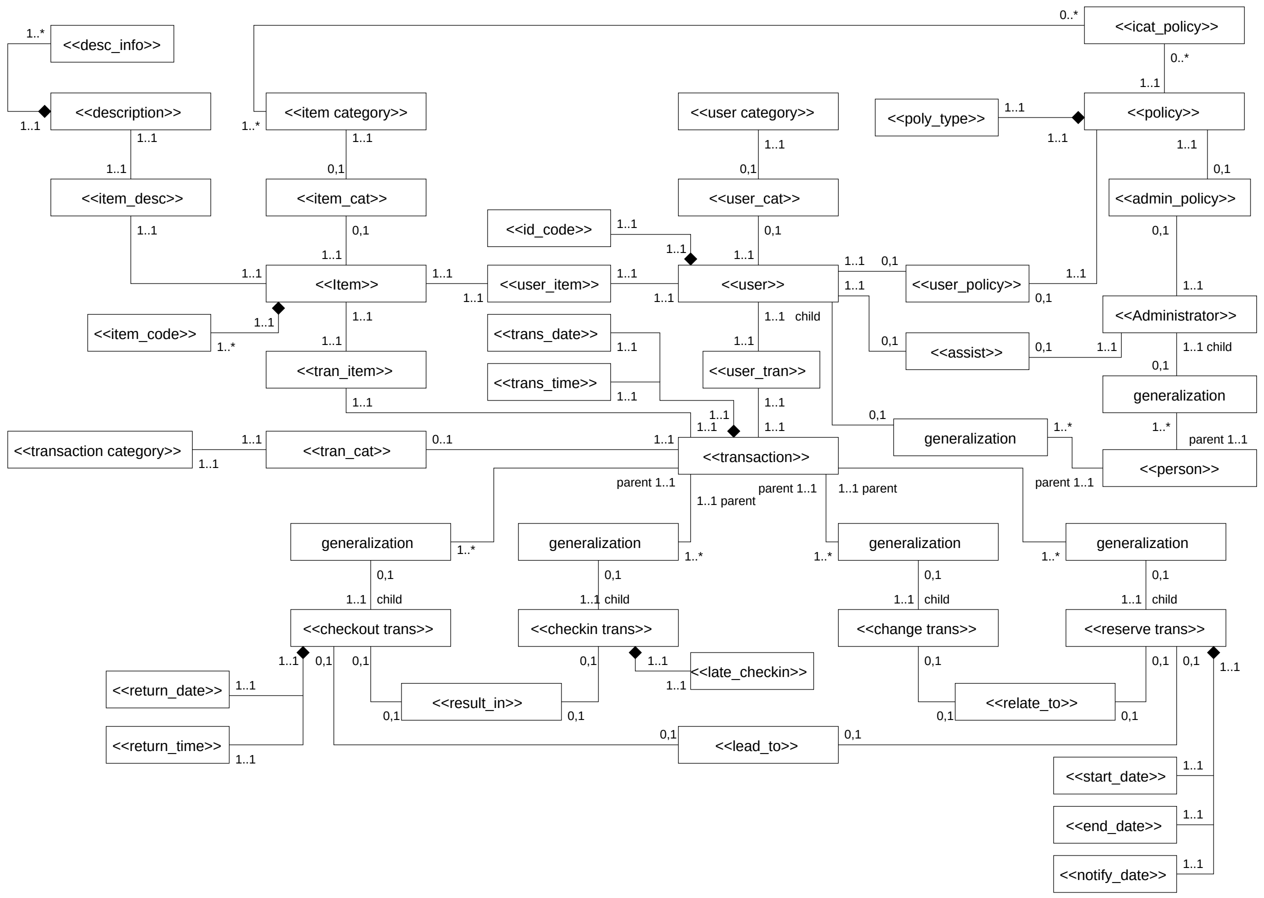

Fig. 4. Meta-Model of *Checkin-Checkout* Domain Class Diagram

Finally, the DSML structure of Figure 2 allows the language developer to include only the meta-models that are pertinent for the domain under consideration, thus eliminating unneeded models from the language.

REFERENCES

[1] Christian Hahn. A domain specific modeling language for multiagent systems. In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, pages 233–240. ACM, International Foundation for Autonomous Agents and Multiagent Systems, May 2008.

[2] Bow-Yaw Wang. Modeling and analyzing applications with domain-specific languages by reflective rewriting: A case study. In Proceedings of the 2006 ACM symposium on Applied computing, pages 1773–1778. ACM SIGAPP, ACM, April 2006.

[3] Scott Thibault, Renaud Marlet, and Charles Consel. A domain specific language for video device drivers: From design to implementation. IEEE Transactions on Software Engineering, 25(3), May/June 1999.

[4] ISO/IEC 19501, Information Technology - Open Distributed Processing,: Unified Modeling Language (UML) Version 1.4.2 (2005).

[5] David Weiss. Commonality analysis: A systematic process for defining families. In Proceedings of the 2nd International Workshop on Development and Evolution of Software Architecture for Product Families, 1998.

[6] Ben Potter, Jane Sinclair, and David Till. An introduction to formal specification and Z, 2nd Ed. Prentice Hall, International Series in Computer Science, 1996.

[7] Tony Clark, Andy Evans, Stuart Kent, Steve Brodsky, and Steve Cook. A feasibility study in rearchitecturing UML as a family of languages using a precise OO meta-modeling approach. Technical Report version 1.0, pUML Group, Sep 2000.

[8] Arie van Deursen and Paul Klint. Little languages: Little maintenance? In Fourth International Conference on Foundations of Object-Oriented Languages. Paris, France, 1997.

[9] Robert B. France, Emanuel S. Grant, and Jean-Michel Bruel. UMLtranZ: An UML-based rigorous requirements modeling technique. Technical report, Colorado State University, Ft. Collins, Colorado, January 2000.

[10] Tony Clark, Andy Evans, and Stuart Kent. Using profiles to re-architect the UML. In Third International Conference on the Unified Modeling Languages (UML2000). York, England, October, 2000.

[11] Tony Clark, Andy Evans, Stuart Kent, and Paul Sammut. The MMF approach to engineering object-oriented design languages. In Workshop on Language Description, Tools and Application. Dipartimento di Informatica e Scienze dell'Informazione (DISI) Universit`a di Genova, April 2001.

[12] Emanuel S. Grant, Ph.D. Dissertation: Defining domain-specific object-oriented modeling languages as UML profiles, Colorado State University, Ft. Collins, Colorado, December 2002

[13] David A. Schmidt. Denotational Semantics: A Methodology for Language Development. Allyn and Bacon, Inc., Newton, Massachusetts, USA, 1986

[14] Desmond F. D'Souza, Aamod Sane, and Alan Birchenough. First class extensibility for UML - Packaging of profiles, stereotypes, patterns. Second International Conference on the Unified Modeling Language ≪ UML ≫'99, Lecture Notes in Computer Science, pages 265–277, Ft. Collins, Colorado, USA, October 1999. Springer-Verlag.

[15] Steve Cook, Anneke Kleppe, Richard Mitchell, Bernhard Rumpe, Jos Warner, and Alan Cameron Wills. Defining UML family members using prefaces. In Mingins C and Meyer B, editors, Technology of Object-Oriented Languages & Systems. IEEE, 1999. Gonsalves and K. Itoh, "Multi-Objective Optimization for Software Development Projects," in *Lecture Notes in Engineering and Computer Science: International Multiconference of Engineers and Computer Scientist 2010,* pp. 1–6.