Formal Methods and Model Transformation Framework for MDA

Gundars Alksnis*
gundarsalksnis@acm.org

Abstract: The article discusses formalization aspects of model transformations in MDA and presents solution from formal specification and category theoretic view. The article gives short overview of the specification language OBJ for the Maude environment, summarizes its properties and discusses common application domains. The introduced extended framework demonstrates how this language can be integrated into method for specifying UML diagrams and further transformations into platform specific system models. The category theoretic orientation is chosen as possible solution to this problem. We apply categorical constructions on formal specification languages, to describe every refinement and transformation step of each specification module. To demonstrate proposed approach a simple example of the distributed software services domain is presented.

Key words: Formal specification, category theory, MDA, model transformation

1 Introduction

The OMG's Model Driven Architecture (MDA) [1] is software engineering vision, in which development process is viewed as a design, refinement and transformation of system models – from abstract to specific ones, from which executable artifacts can be automatically generated.

Although MDA's approach seems promising to software engineering industry and already has practical experience and success stories in commercial projects, however, from the formalization perspective this approach still seems too vague. In this approach, automated transformations of Unified Modeling Language (UML) models play a key role. There is ongoing progress on MOF 2.0 Query/Views/Transformations standardization [2]. However, our opinion is that there is not sufficient formal theoretical background. On the other hand, development with formal methods has much longer history and various formal specification languages have been rigorously applied in software engineering.

There already have been made attempts to describe MDA model transformations by means of UML meta-model [3]. However, we think, they are mainly oriented on particular aspects of transformations (e.g. design patterns recognition [4]) and cannot be universally applied.

On the other hand, if we are able to translate graphical models into symbolic representations, they would be more appropriate for computer processing – for example, most formal specification languages, supports not only semantics (like XML), but also functional relationships between language elements – axioms and theorems. We can then operate with those symbols (e.g. use theorem provers) to verify specifications and corresponding model transformations.

Therefore, the main problems to solve are (1) development of appropriate model transformation into symbolic representation; (2) transformation of the symbolic representation

*Department of Applied Computer Science, Riga Technical University, Meza 1/3, Riga, LV-1048, Latvia

into desirable target representation; and (3) transformation to the new, refined, model (which is formally derived from the initial model). We think that this process should be implemented in the software tools. This will eliminate transformation errors and increase model transformation efficiency.

The category theoretic orientation has been chosen as a one possible solution to this problem [5]. It is because there already have been made developments in the transformation of formal specifications by means of categorical morphisms and operations [6]. When we apply categorical constructions on formal specification languages, we can formally describe every refinement and transformation step of each module specification.

The article gives short overview of the specification language OBJ in the implementation for the Maude environment, summarizes its properties and discusses common application domains. Afterwards, initial description of transformation framework demonstrates how this language can be integrated into method for specifying UML diagrams and further transformations into platform specific system models. To demonstrate proposed approach, a simple example of distributed software services domain is presented. Finally, conclusion gives evaluation of our method and suggests avenues for further research.

2 Formal Specification with OBJ

The algebraic specification language OBJ initially was designed in 1976 by J. Goguen [7]. Nowadays, with the term OBJ refers to the language family, which includes many its variants and developments. Latest version of this specification language is named OBJ3. One of its implementation – the Maude environment, has been developed by SRI International [8]. In it OBJ3 is extended with rewriting logic and has been successfully applied in meta-programming, reflection, and algorithm implementation. In addition, the Maude environment has been chosen for demonstration of proposed framework.

This language supports algebraic programming – specification of abstract data types and operations on them. It is based on order sorted equational logic and is enriched with other logics (e.g. rewriting, hidden equational or first order logic), and provides the powerful module system of parameterized programming. Among other things, it has been successfully used for research and teaching in software design and specification, rapid prototyping, theorem proving, user interface design, and hardware verification.

Specification in Maude consists of module definitions and relations between them – new modules may include or extend available modules. There are two separate levels of Maude: Core and Full Laude. The Full Maude fully includes Core Maude by has additional support for object-oriented (OO) modules. OO module is declared with the following syntax [9]:

```
(omod <NAME> is ... endom)
```

The ellipses denote all the declarations and statements for this module. All statements must end with the period. To describe algebra one declares sorts and the operations (ops) and class messages over these sorts.

Classes in Maude are declared with a class name together with attributes linked to objects of that class. Attributes are declared with an attribute name and the attribute's sort. For example, the class of table might be defined as:

```
class TABLE | occupied : Bool , chairs : Nat .
```

This class defines two attributes – `occupied` of Boolean type and `chairs` as a natural number.

Messages are specific versions of operations. They are declared with the key word `msg`, and after `msg` comes the name (in either prefix or mixfix notation), followed by a colon, then the sorts fed into the message, then the `->` symbol. The two special requirements are that, for

messages, the -> symbol is always followed by the key sort Msg, and the first argument is always an Oid (object identifier). For example:

```
msg sit : Oid -> Msg .
```

Another essential aspect of Full Maude is that it supports parameterization. The idea of parameterization is that one creates a parameterized sort (template), and a parameter sort, which plugs into the parameterized sort and creates a useful combination. The first requirement for a parameterized anything is a theory. A theory sets the rules for a parameter, a sort of schematic, which the parameter must fulfill in order to be accepted by a parameterized sort. The trivial and most common theory TRIV is already included in the Full Maude:

```
(fth TRIV is sort Elt . endfth)
```

This theory is a functional theory – it is for the benefit of functional modules. To define the actual parameterized module, we declare it with a special notation:

```
(fmod EXAMPLE(X :: TRIV) is ... endfm)
```

The (X :: TRIV) phrase declares that there is one or more parameterized sort in the module (with the label X) whose parameter must fulfill the theory TRIV. Instead of X, we can use any letter or identifier, as long as we are consistent, however, the convention is a single capital letter, usually X.

When we have the theory and the parameterized module, we have to fulfill next requirement – a module that defines the parameter. To link all three concepts together, we define a view. The view maps the sorts and operations defined in the theory to those in a given parameter. This establishes a link between the two, which allows the Maude to understand the parameter sorts as parameters. We create a view using the special syntax:

```
(view <ViewName> from <THEORY> to <MODULE> is ... endv)
```

Now, then we have outlined main principles and syntactic elements, we are ready to turn to the proposed transformation integration framework.

3 Integration with MDA by Means of the Category Theory

Motivation of our integration method was inspired from [10], in which M. Bujorianu describes method, which uses heterogeneous partial (also called viewpoint) specification integration.

Method of UML graphical model translation to formal specifications is discussed in [11]. In it, authors demonstrate approach, where code generated from UML diagrams can be formally verified. They accomplish it by formalizing UML meta-model level constructs in the formal specification language Slang.

The purpose of their research is to bridge formal and tool-based development methods by deriving a methodical process for checking the correctness of the automatic translation of a UML diagrams to a formal language with respect to the UML meta-model. This process consists of formalization of a subset of the UML meta-model and of showing that the class of models generated by the translator is within the class of models of the UML meta-model.

The formalization of both UML, and UML application specifications, must be performed in an extensible and computerized fashion that supports specification composition and tool interoperability. The focus of their research is in the meta-modeling and validation process and generalizable tool support to the formalization of many semi-formal modeling languages.

Figure 1 portrays their UML formalization process approach, but it is extended with our method of integration in the MDA. It is adopted for the specification language OBJ3 in the implementation of Maude, but may be extended to any algebraic/category theoretic specification language (e.g. Z, TLA+, Slang, etc.).

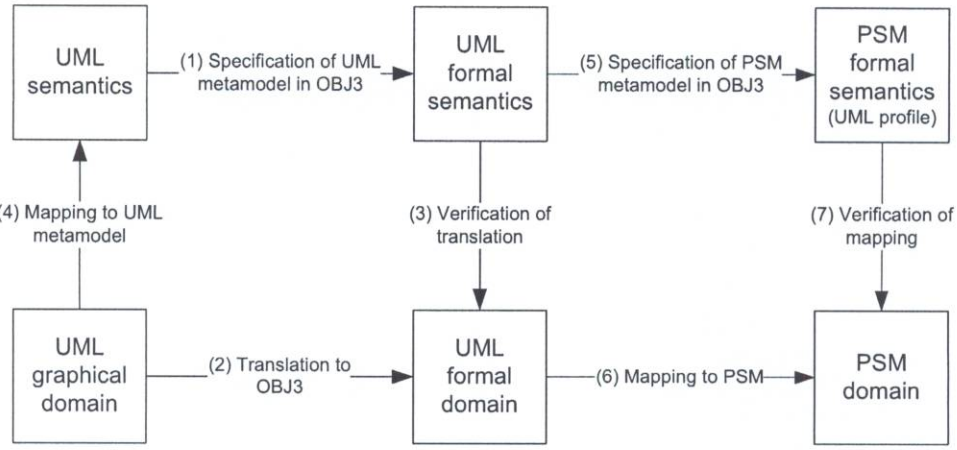

Figure 1. UML model translation and verification scheme.

Shortly, depicted mappings describe the following. Mapping (1) from 'UML semantics' to 'UML formal semantics' describes the formalization of the UML meta-model. Formalization rules describe how to translate UML semantics meta-objects into OBJ3, with a rationale for each formalization rule and a description of formalization rule alternatives.

Mapping (2) from 'UML graphical domain' to 'UML formal domain' shows the software and tool support needed to translate UML applications (described as the 'UML graphical domain') to OBJ3, according to the formalization rules. The OBJ3 version of the 'UML graphical domain' is called the 'UML formal domain'.

Mapping (3) from 'UML formal semantics' to 'UML formal domain' represents the verification of the correctness of the translation. Here, the instances of the OBJ3 from the 'UML graphical domain' are checked for compatibility with the OBJ3 representation of abstract UML theory ('UML formal semantics' node). Instances are checked to ensure that the OBJ3 from the 'UML graphical domain' view preserves the UML semantics captured in the mapping (1) between 'UML semantics' to 'UML formal semantics'. To explain this step, both, the specification of the UML meta-model, and of any UML diagram, are viewed as a presentation of OBJ3 parameterized module. The goal is to ensure that the class of models of the theory, obtained as a translation of any UML diagram, is a sub-sort of models of the theory of the UML meta-model. In order to show this, it must be shown, that for each such translation, a view can be obtained from the UML meta-model theory to the theory representing a given UML diagram.

Mapping (4) from 'UML graphical domain' to 'UML semantics' is a mapping of all possible UML diagrams that can be produced within a UML tool into the UML meta-model. In other words, this mapping represents an interpretation of a UML diagram in the UML meta-model. Mappings between 'UML semantics' to 'UML formal semantics' and mapping between 'UML graphical domain' and 'UML semantics' collectively constitute the formalization of UML diagrams.

Our proposed extension is shown in the remaining arrows and edges. Here, the mapping (5) from 'UML formal semantics' to 'PSM formal semantics' is a mapping to MDA's Platform Specific Model (PSM) meta-model, which, for example, may be UML profile or transformation model. This mapping is specified in a formal specification language. Knowing specification mapping semantics, we can map formal specification from 'UML formal domain' to 'PSM domain' in a precise manner (mapping (6)) by using appropriate

transformation rules, and afterwards verify results with the requirements of 'PSM formal semantics' (mapping (7)).

4 Transaction Specification Problem

To demonstrate proposed approach, this section discusses the problem of distributed transaction example specification.

Solutions of this problem may be applied in the domains where Database Management System's (DBMS) data locking sub-system is not enough matured for use in the client applications, which usually operates at the business process levels. For example, if user opens for editing some data set which are not trivial and involves many tables simultaneously (for example, invoice document), there may not be simple way to lock all such information at the database level, where usually only current records are locked (to minimize concurrency). So there should be implemented some kind of semaphore system which does not depend on DBMS and may be applied to distributed client-server applications. Another solvable problem is how to implement database level notifications, when changes of some global data must be notified to every currently active client application for cached data refresh requirement. In fact, such system could not be applied only to databases, but everywhere where synchronization and locking mechanism is applicable.

Informal requirements for this problem are the following:

- At the beginning client establishes permanent connection with the service and registers itself by supplying its information (e.g., client name, application, mode within the application, etc.);
- After connection have been successfully established, client may perform the following actions:
  - Register to notification(s);
  - Trigger one of the registered notification;
  - Check if some registered notification has been triggered;
  - Unregister to notification(s);
  - Try to acquire semaphore – if acquisition was successful (no one else has blocked it), server blocks particular semaphore for this client. While it is blocked, other attempts to this semaphore (either from the same or other client) must fail, by returning to the client information about blocking client;
- Perform temporary connection functions, like:
  - Request current service state (must return information about all currently active connections, notifications and semaphores);
  - Forcefully disconnect client connection and release its semaphores and notifications (e.g. in case of client application failure or crash);
  - Open, read, modify or close special remote registry service where all clients can share common data;
- Service operates in reactive mode, i.e. only clients can request actions to be performed;
- Data sent between client and service consists of fixed size data packet which contains following fields:
  - Code of the operation to be performed;
  - Flag indicating whether more packets to be followed;
  - Data buffer of constant size for textual and/or binary data (may be only partially fulfilled);
  - Four data fields for integer information;

- and, operation result code;
- Optionally, data packets can be encrypted and/or CRC checked for security and validity reasons.

Based on given informal requirements, Figure 2 depicts corresponding class diagram, which afterwards are translated into formal specifications.

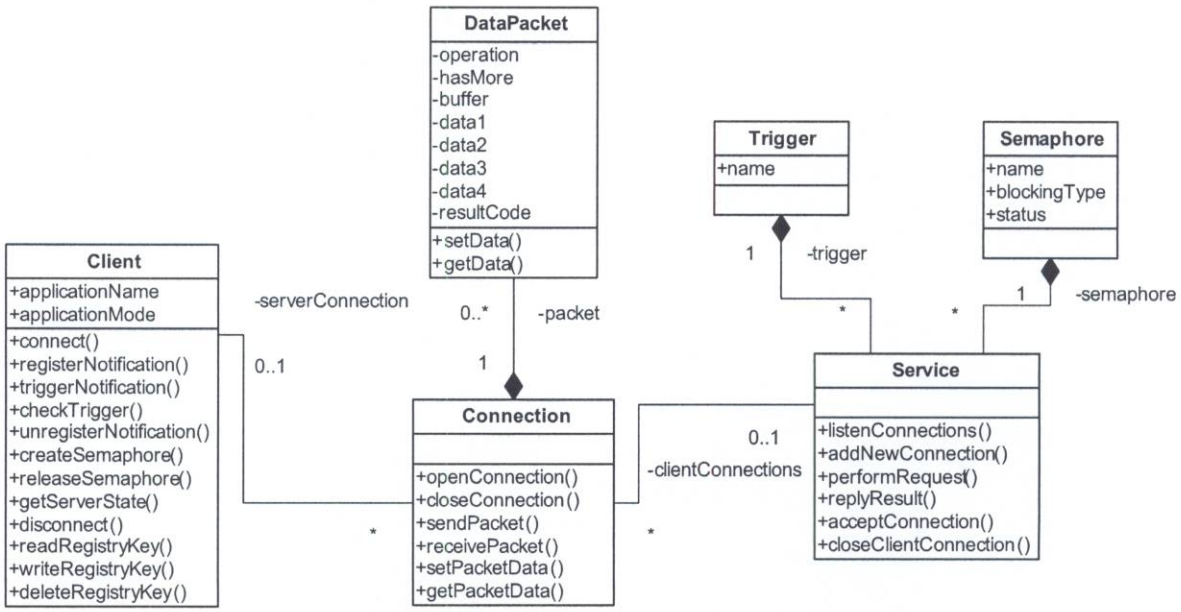

Figure 2. Sample distributed transaction application UML class diagram.

To illustrate corresponding formal specifications for the Maude environment, the two classes' specifications – *DataPacket* and *Connection*, are shown in the Figure 3 and Figure 4 respectively.

As may be seen from the specifications, they are only initial class specifications. There should be added equations, axioms and theorems, which proves validity of the specification.

5 Evaluation

Designed formal specifications were checked in the Maude environment against errors and inconsistencies. Main objective of mentioned two specifications is to show that corresponding formal specifications can be obtained from the graphical UML diagram.

In this method, each specification is considered as independent object, but morphisms between them show how these objects are interconnected. This allows specifying particular aspects of the system in the specification languages, which has better support to expressions of such properties. For example, one specification object can specify processes whilst other specifies data structures and/or states.

This method can be applied to the proposed MDA model transformation framework where graphical models are transformed into formal specification languages and after appropriate refinements, transformed to the new representation to form refined platform specific models. Proposed approach can assure preservation of consistency and modularity. Moreover, it can be automated in the software tool, in which manipulations with symbolic information are more efficient than graphical.

```
(omod PACKET is
   class Packet | operation : Operation
                , hasMore : Bool
                , data1 : Int
                , data2 : Int
                , data3 : Int
                , data4 : Int
                , resultCode : Int .
   ops connect
       disconnect
       registerNotification
       triggerNotification
       checkTrigger
       unregisterNotification
       acquireSemaphore
       releaseSemaphore : -> Operation .
   ops failed succeded : -> ResultCode .
   msg setData : Oid Operation Bool String Int Int Int Int ResultCode
                 -> Msg .
   msg getData : Oid -> Msg .
endom)
```

Figure 3. Formal specification of *DataPacket* class.

```
(omod CONNECTION is
   class Connection | packet : Oid .
   msg openConnection : Oid String String Int -> Msg .
   msg closeConnection : Oid -> Msg .
   msg sendPacket : Oid -> Msg .
   msg receivePacket : Oid -> Msg .
   msg setPacketData : Oid Operation Bool String Int Int Int Int ResultCode
                       -> Msg .
   msg getPacketData : Oid Packet -> Msg .
endom)
```

Figure 4. Formal specification of *Connection* class.

Currently we are at the initial step of such transformation implementation. In the future, we plan to extend our method with the partial specification (e.g. viewpoint, as in [10]) implementation, in which each model is specified and checked in different specification language (which is the most appropriate for the task) and then, by using specification integration, each particular model specification is integrated into common system model.

To accomplish this, for each specification language *n* the category SPEC*n* will be developed – objects constitute specifications in that language and morphisms specify refinements between them. Each specification object in this category consists of the following parts: vocabulary, which describes the sorts, constants, attributes, actions; and behavior, which describes realization in terms of temporal logic axioms, predicates or actions, depending on used specification language. It ensures coherent and well-founded theoretical basis for representing structure in formal specification languages. Each category SPEC*n* must be finitely complete and co-complete, i.e., it must have initial and terminal objects and there must exist pullbacks and pushouts. These properties may be automatically proven. By analyzing morphisms of those categories, appropriateness for model representation may be evaluated.

In addition, categories PIM and PSM should be studied. In those categories, objects are models (e.g. UML) and morphisms show how those models are interconnected. Afterwards, the functors between categories PIM and PSM will be researched. After that, we are planning to research morphisms from model representation (e.g. in XML and XMI) to symbolic representation on which categorical operations later may be applied.

6 Conclusion

In the article, we outlined method to model transformations from the view of formal specification languages and categorical constructions application to specifications from graphical UML models to PSM models with the ability of rigorous validation of each transformation step.

Outline of specification language OBJ3 where discussed and simple example demonstrated formal representation of partial class diagram in the domain of distributed transaction processing.

After evaluation of our approach, we sketched future research avenues of our method.

This work has been partly supported by the European Social Fund within the National Programme "Support for the carrying out doctoral study program's and post-doctoral researches" project "Support for the development of doctoral studies at Riga Technical University".

References

1. OMG Model Driven Architecture. http://www.omg.org/mda/
2. T. Gardner, C. Griffin, J. Koehler, and R. Hauser. "A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the Final Standard", 2003. http://www.omg.org/docs/ad/03-08-02.pdf
3. R. Sheena, S. Judson, R. France, and D. Carver. "Specifying Model Transformations at the Metamodel Level", 2003. http://www.metamodel.com/wisme-2003/19.pdf
4. D. Lopes, S. Hammoudi, J. Bezivin, and F. Jouault. "Mapping Specification in MDA: From Theory to Practice", 2004. http://interop-esa05.unige.ch/INTEROP/Proceedings/Interop-ESAScientific/PerPaper/I08-2%20354.pdf
5. G. Alksnis. "Formal Specification from Category Theory Viewpoint", Computer Science, Applied Computer Systems, Vol.17, Nr.5, Scientific Proceedings of Riga Technical University, Riga, 2003, pp.137-144.
6. Specware. 2006. http://www.specware.org/
7. J. Goguen, T. Winkler, J. Meseguer, et al. "Introducing OBJ – Applications of Algebraic Specification using OBJ", 1993. http://www-cse.ucsd.edu/users/goguen/ps/iobj.ps.gz
8. The Maude System. 2005. http://maude.cs.uiuc.edu/
9. M. Clavel F. Duran, S. Eker, et al. "Maude Manual (Version 2.1.1)", 2005. http://maude.cs.uiuc.edu/
10. M. C. Bujorianu . "Integration of Specification Languages Using Viewpoints", Integrated Formal Methods, 4th International Conference, IFM 2004, Lecture Notes in Computer Science, Springer, 2004, pp. 421 - 440.
11. J. Smith, M. Kokar, and K. Baclawski. "Formal Verification of UML Diagrams: A First Step Towards Code Generation", pUML 2001, Toronto, Ontario, Canada, 2001, pp. 224 -240.