Behavior Model Mapping and Transformation using Model-Driven Architecture

Mohammed Abdalla Osman Mukhtar*, Azween Abdullah**, and Alan Giffin Downe***

Abstract—Model mapping and transformation are important processes in high level system abstractions, and form the cornerstone of model-driven architecture (MDA) techniques. Considerable research in this field has devoted attention to static system abstraction, despite the fact that most systems are dynamic with high frequency changes in behavior. In this paper we provide an overview of work that has been done with regard to behavior model mapping and transformation, based on: (1) the completeness of the platform independent model (PIM); (2) semantics of behavioral models; (3) languages supporting behavior model transformation processes; and (4) an evaluation of model composition to effect the best approach to describing large systems with high complexity.

Keywords—MDA; PIM; PSM; QVT; Model Transformation

I. INTRODUCTION

Despite the growth of interest in model-driven architecture (MDA) there is still little agreement on how behavioral aspects should be supported with the approach. Considerable effort has been devoted recently to model mapping and the transformation from platform independent models (PIMs) to platform specific models (PSMs) in many application domains. Much of this work has focused heavily on behavioral aspects of PSMs. There is a need for broader consideration of behavior model mapping using either vertical mapping (refinement) or horizontal mapping (from PIM to PSM).

The central idea of Model Driven Architecture (MDA) which sponsored by Object Management Group (OMG) is that developer should develop models, not programs. That is not to privilege a graphical over a textual programming, but rather to make the developer to be enabled to work at as a high level of abstraction especially in the PIM to add more details to make the service specification behavior is refined (see figure1) may become hard to guarantee in a single refinement transformation, T1. Therefore, an intermediate step in which correctness and consistency particularly of behavioral issues must be addressed in the refinement transformation. However, when trying to realize this refinement transformation, the gap between service specification and platform-independent service design can become rather wide, such that correctness and consistency becomes hard to guarantee in a single refinement transformation, T1. Therefore, an intermediate step in which the service specification behavior is refined (see figure1) may be necessary.

II. COMPLETENESS OF PIM

If one traces the development of MDA approaches, it can be seen that most research has focused attention on: (1) structural aspects of the PSM level; and (2) processes for generating code. Much less attention has been typically devoted to the PIM model level or to the behavior of modeled applications. One exception was found in the work of Daniele et al. [3], which presented an MDA-based approach that incorporated behavior modeling at the PIM level, but within a specific category of applications. These authors argued that behavior in a PIM can be divided to more than one layer of abstraction, the first one being more independent than subsequent layers, and the deeper ones, essentially, moving nearer to a PSM.

This approach was applied to a Mobile System (M-MUSE DSL), in which the platform-independent design phase was decomposed in the service specification and platform-independent service design steps [3]. PIM design, it was argued, should be a refinement of the service specification, which implies that correctness and consistency particularly of behavioral issues must be addressed in the refinement transformation. However, when trying to realize this refinement transformation, the gap between service specification and platform-independent service design can become rather wide, such that correctness and consistency becomes hard to guarantee in a single refinement transformation, T1. Therefore, an intermediate step in which the service specification behavior is refined (see figure1) may be necessary.
III. SEMANTICS OF BEHAVIORAL MODELS

Key to the development of the MDA approach has been the extensive work aimed at defining the description semantics of Object Constraints Language (OCL), the programming or modeling languages. For that, metamodelling, since the beginning of this decade, has become a widely used tool to describe the abstract syntax of modeling languages. There are two generally acknowledged approaches to describing OCL constraints semantics [4]. For instance, this constraint eval:

$$\text{CONSTRAINT} \times \text{STATE} \mapsto \{\text{true}, \text{false}, \text{undefined}\}$$

Can be defined either mathematically by using structural induction over CONSTRAINT (refer to [5]), or logically like using Isabelle/High-Order Logic (HOL).

These two approaches have a good manner to evaluate OCL constraints in a formal and non-ambiguous method, but they still have some disadvantages. First disadvantage is this gap between OCL’s official syntax definition which is given as metamodel, and the OCL’s syntax which is given in structural induction. Second, which is the main drawback is the understandability.

The main technique to heal the rift of this gap and to get good understandability is metamodelling. Metamodels are already used to define abstract syntax with very expressive and easy to understand. It is already used to define the semantics of class diagrams. This technique is sponsored by OMG using Evaluation-Metaclasses [4], and this approach is provided using transformation rules written in QVT. Figure2 shows metamodel for OCL abstract syntax, and figure3 shows metamodel for the semantics of OCL. In [6] also applying graph transformation to OCL constraints semantics.

Now, in recent years, OCL becomes a constraint language that is applied to various modelling languages, instead of just it is a language used to constrain UML models. This includes Domain Specific Languages (DSLs), and meta-modelling languages like MOF or Ecore.

The new trend is going on providing variability to OCL parsers to work with different modeling languages; variability concentrate on the technical space which models are implemented in (like Java, Ecore, or a specific model repository). In [7] the authors argued that all OCL tools support variability at the model level (OCL compilers), for that they said we can support variability at the model instance level (OCL interpreter) and proposed a generic adaptation architecture for OCL interpreters that hides models and model instances behind well-defined interfaces. This enables reuse of the complete OCL infrastructure including the OCL parser, standard library and interpreter. There is also some work done for modeling operational semantics of domain specific modeling language (DSML) as presented in [8], which
applied this approach to petri nets as well as for a stream-oriented language in the domain of earthquake detection.

IV. SUPPORTING LANGUAGES FOR MAPPING OF BEHAVIOR MODELS

There is a lot of transformation languages working as a tool to make the transformation operation full automated, we have chosen a three types of these languages depend on some criterion. First one is Query, View, Transformation (abbreviated by QVT) which is most standardized, which is sponsored by Object Management Group (OMG). The second one is KerMeta (abbreviation of Kernel Metamodel), it is domain specific language, it is building basically on Object Oriented Programming, and it can be plugged on Eclipse. The third one is MATA (abbreviation of Modeling Aspects using a Transformation Approach), from its’ name we can see that it is building on Aspect Oriented Programming. Now we need to take each language individually, and focusing the light on some its’ features, and making technical comparison.

A. QVT

QVT (Query/Views/Transformation) is the OMG standard language for specifying model transformations in the context of MDA. It is regarded as one of the most important standards since model transformations are proposed as major operations for manipulating models [8]. The three concepts that are used in the name of the QVT language as defined by OMG documents are: [9]

Query: A query is an expression that is evaluated over a model. The result of a query is one or more instances of types defined in the source model, or defined by the query language.

View: A view is a model which is completely derived from another model (the base model). There is a ‘live’ connection between the view and the base model.

Transformation: A model transformation is a process of automatic generation of a target model from a source model, according to a transformation definition.

QVT languages are arranged in a layered architecture shown in Figure 4. The languages Relations and Core are declarative languages at two different levels of abstraction. The specification document defines their concrete textual syntax and abstract syntax. In addition, Relations language has a graphical syntax. Operational Mappings is an imperative language that extends Relations and Core languages. Relations language provides capabilities for specifying transformations as a set of relations among models. Core language is a declarative language that is simpler than the Relations language. One purpose of the Core language is to provide the basis for specifying the semantics of the Relations language. The semantics of the Relations language is given as a transformation RelationsToCore. This transformation may be written in the Relations language.

Sometimes it is difficult to provide a complete declarative solution to a given transformation problem. To address this issue the QVT proposes two mechanisms for extending the declarative languages Relations and Core: a third language called Operational Mappings and a mechanism for invoking transformation functionality implemented in an arbitrary language (Black Box implementation).

B. KerMeta

KerMeta is a meta-language for specifying the structure and behavior of models. It has been also been developed as a core language for Model Driven Engineering (MDE) platform. KerMeta is an executable metamodeling language implemented on top of the Eclipse Modeling Framework (EMF) within the Eclipse development environment. Figure 5 shows three main windows in KerMeta Graphical Interface. The first one is the metamodel using class diagram (which is a subset from UML class diagram MOF metamodel), the second widows is the KerMeta code to describe the class diagram, and the last one is the summarization for the class diagram.

C. MATA

MATA takes a different approach to aspect-oriented modeling (AOM) since there are no explicit join points. Rather, any model element can be a join point, and composition is a special case of model transformation. The
Aspect model composition is another approach of combining two models, MB and MA, where an aspect model MA is said to crosscut a base model MB. As such, aspect model composition is a special case of the more general problem of model fusion. A number of techniques and languages have been developed to specify how MA crosses MB, and, in particular, how MA and MB should be composed [12].

VI. CONCLUSION

In this paper we are focusing on behavior model transformation in order to push the wheel of behavior model transformation development, and to be aware about some aspects that we can contribute on to participate in these developing. These aspects are Completeness of Platform Independent Model (PIM), Semantics of Behavior Models, Supporting Languages for Mapping of Behavior Models, and Model Composition.

REFERENCES