ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.8, August 2006, pp.1698−1706
DOI: 10.1360/jos171698
© 2006 by *Journal of Software*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于动作语义的 UML$_{1.X-2.0}$模型转换方法[*]

陈秀红[+], 何克清, 何璐璐

(武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

An Action Semantic-Based Approach to UML$_{1.X-2.0}$ Model Transformation

CHEN Xiu-Hong[+], HE Ke-Qing, HE Lu-Lu

(State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

+ Corresponding author: Phn: +86-27-68754590, E-mail: your-hong@163.com

Chen XH, He KQ, He LL. An action semantic-based approach to UML$_{1.X-2.0}$ model transformation. *Journal of Software*, 2006,17(8):1698−1706. http://www.jos.org.cn/1000-9825/17/1698.htm

Abstract: UML2.0 standard has been adopted by the OMG for a long time. However due to the popularity of UML1.X in industry, a huge number of practical applications and models based on UML1.X still exist, and they are no longer an accurate description of the systems under the UML2.0. Today, many tools support the modeling with UML2.0, but none of them supports the transformation from UML1.X models to UML2.0 models. This paper compares the differences between the two versions from the top-level of metamodel, chooses a declarative and imperative hybrid framework, and presents a UML model transformation method based on Action Semantic. It also describes the transformation of the Interaction metamodel with Action Semantic Language, which proves the feasibility of the approach. This approach can reduce the repeated work of the user and realize the reuse of software models, and can also be applied to other metamodel or model transformations.

Key words: model-to-model; model transformation; interaction; action semantic

摘 要: 虽然UML2.0标准被OMG组织采纳已久,但由于UML1.X标准在工业界的广泛使用,仍然存在大量的实用模型和应用软件,它们在UML2.0标准之下已不能准确描述系统.UML2.0建模工具并不支持从UML1.X模型到UML2.0模型的转换.从顶层元模型的角度比较了这两个版本的不同,采用声明式和命令式混合的模型框架,给出了一种基于动作语义的UML模型转换方法,并用ASL描述交互元模型的转换实例,验证了方法的可行性.该方法将减少用户的重复劳动,实现软件的模型重用,也适用于其他元模型或模型层次上的转换.

关键词: 模型-模型;模型转换;交互;动作语义
中图法分类号: TP309 文献标识码: A

[*] Supported by the National Natural Science Foundation of China under Grant No.60373086 (国家自然科学基金); the Wuhan Science & Technique Key Project Foundation of China under Grant No.20021002043 (武汉市科技重点项目) ; the Natural Science Foundation of Hubei Province of China under Grant No.2002ABB037 (湖北省自然科学基金); the Open Foundation of SKLSE of China under Grant No.03-03 (软件工程国家重点实验室开放基金)

Received 2004-12-23; Accepted 2005-05-31

1 Introduction and Background

UML (unified modeling language) has evolved rapidly since its creation. Before the emergency of UML2.0, UML has improved its modeling power and precision but has not changed in its essential nature. In October 2003, the OMG (object manage group) adopted the UML 2.0 standard, with the aim, among others, to provide a solid base for MDA (Model-Driven Architecture) via a simplified mechanism for creating profiles.

UML 1.4 was the culmination of a concerted effort to create a practical tool for modeling systems. The focus was on “practical”. The tool had to be useful for a broad spectrum of projects and easily implemented by users and tool vendors alike[1]. UML 2.0 provided the opportunity to go back over the tool and fine-tune the definitions, clean up the architecture, and generally refine the tool to ensure its long-term success. Also, while UML was gaining acceptance, OMG was hard at work promoting MDA. UML is a major component of MDA, so complete alignment with the other elements of MDA is essential[1].

Due to UML2.0 and the popularity of UML1.X in industry, a huge number of applications and models that are based on UML1.X exist. When domain evolution is required, the obsolete models created with the UML1.X are no longer an accurate description of the systems with respect of the UML2.0. In spite of this, these legacy models and applications are still great assets for us. Most of them are still used in practice. Abundant domain knowledge and design knowledge are contained in these legacies. To avoid rebuilding everything from scratch in the evolved modeling paradigm, it is necessary that these knowledge asserts should be migrated into the new modeling environment to continue their value.

There are many CASE tools claiming on the supporting of UML2.0: Telelogic’s TAU, iLogix’s Rapsody, Artisan’s Real-Time Studio, MetaMatrix MetaBase Modeler, Sparx Systems Enterprise Architect, and so on. But none of them have capability to reuse the UML1.X legacy models. We think the main reason is that there hasn’t a standard specification to specify the model transformation. In order to make their products widely used and have strong compatibility, most of them are waiting the issuance of MOF2.0 Query/Views/Transformations (QVT) Specification by OMG[2].

For those reasons, the research on the issue of UML1.X to UML2.0 model transformation is very necessary and pressing. In our project, we first compare the disparities between the corresponding elements in UML1.X and UML2.0, including the package they belong to and the metamodel of themselves. Then, we define the model transformation of the two versions, and our primary work of realizing the UML1.X to UML2.0 model transformation based on the metamodels is to establish the mapping relation from the UML1.X to the UML2.0. As the mapping of UML1.X to UML2.0 is defined in terms of syntax patterns, it is possible to use pattern recognition to transform sub-graphs of the existing domain models into the sub-graphs in the evolved domain so that their mapping into the semantic domain is semantically correct. Therefore, we should recreate the existing domain models using a generated transformation, rather than human labor. Finally, we describe the model transformation with Action Semantic Language (ASL) which is consistent with the UML Action Semantics. It is hoped that it can impulse the birth of the needed tool.

This paper is organized as follows: Section 1 introduces the difference between UML1.X and UML2.0, and based on that, identifies the problems we are facing. Section 2 gives an overview of the difference between UML1.X and UML2.0 from the top-level of metamodel. Section 3 analyzes the existing transformation approach and describes our transformation approach based on Action Semantic, which is the main part of this paper. In Section 4, we provide a case study of our transformation approach on the interaction metamodel of interaction package describes it by ASL. In the last Section, we specify our contribution and conclude with an outlook on the future work.

2 Overview of the Difference Between UML1.4 and UML2.0

2.1 The kernel package

UML1.4[3] consists of three top-level packages and additional specifications for Action Semantics and the Object Constraint Language. Each of the three top-level packages contains a unique set of resources needed to define a UML model. Figure 1 represents the three packages: Behavioral Elements and Model Management packages, which depend on the Foundation package[1]. The Foundation package provides the model with elements that are required throughout the metamodel in the construction of other elements. Figure 2 identifies three sub-packages: core elements, extension mechanisms, and data types. Almost all of the diagram elements in UML derive their basic features from the elements defined in these three packages.

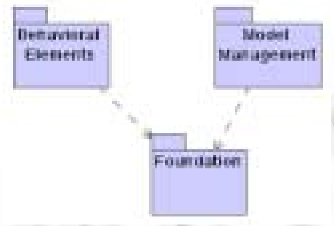

Fig.1 Top-Level packages of UML1.4

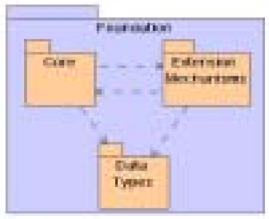

Fig.2 UML1.4 foundation packages

UML2.0 is comprised of two libraries: the infrastructure[4] and the superstructure[5]. The infrastructure library contains the core and profiles packages. The core package is the metamodel at the heart of the MDA architecture. The Profiles package defines the UML extension mechanisms for creating a UML dialect, variations on the basic language that are customized to specific environments or application subject areas. The superstructure extends and customizes the infrastructure to define the UML metamodel.

The core package provides the foundation on which to build the MOF, UML, CWM, Profiles and future languages. Figure 3 shows the relation of them. The core package contains three other packages: abstractions, basic, and constructs. They all depend on the data types defined in the primitive types package. See Fig.4, the primitive types package defines a small set of data types that are used to specify the core metamodel. (Integer, Boolean, String). The abstractions package defines the common concepts needed to build modeling elements such as classifiers, behavioral elements, comments, generalizations, and visibilities. (Almost all of these metaclasses are abstract, meaning that they may not be instantiated.) The basic package defines the common characteristics of classifiers, classes, data types, and packages. The construct package refines contents of the abstractions and basic packages, merging and refining abstract concepts to create a set of common modeling elements[1].

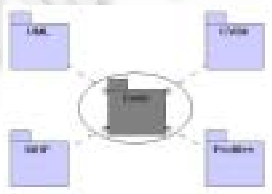

Fig.3 The core metamodel of UML2.0

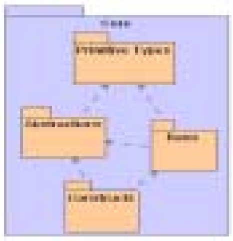

Fig.4 UML2.0 core package

3 Action Semantic Based Model Transformation Framework

3.1 The existing model transformation approach

A transformation generates a target model from a source model[2]. Transformations may be *one-way* (unidirectional) or *two-way* (bi-directional). There are two main approaches to specify the model transformation: imperative approach and declarative approach. The following two definitions have been taken from the Free Online Dictionary of Computing[6]: Imperative (or procedural) languages specify explicit sequences of steps to follow to produce a result, while declarative languages describe relationships between variables in terms of functions or inference rules and the language executor (interpreter or compiler) applies some fixed algorithm to these relations to produce a result. Any programming language that specifies explicit manipulation of the state of the computer system, not to be confused with a procedural language is an imperative language. The most common examples of declarative languages are logic programming languages such as Prolog and functional languages like Haskell.

The authors of Ref.[7] believe that a declarative approach is useful for specifying simple transformations and for identifying relations between source and target model elements. However, many transformations are not straightforward, and for complex rules, the declarative approach would probably have unrealistic computation times or counter-intuitive results. An imperative language is preferable for the definition of complex many-to-many transformations that involve detailed model analysis, but it abstracts level is very low. The standpoint of Adam Bosworth[8] supports their points of view.

3.2 QVT

Model-to-model transformation is a key technology for OMG’s MDA. The need for standardization in this area leads to the MOF QVT RFP form OMG[2]. Since the RFP hopes that submissions could support a hybrid approach to transformation definitions[7], there are many approaches trying to resolve the dilemma of declarative vs. imperative, such as TRL (transformation rule language)[9]. It is defined by normative metamodel and a standard library, and in addition to this, a normative concrete textual syntax is provided. TRL can be viewed as an extension to OCL2.0 since it reuses the navigation facilities that are available in that language. The authors of ref.[10] declare that giving the explicit imperative behavior, taking the pre- and post-conditions of the initial rule as specification contracts for this implementation, and assuming transformation inputs are MOF-compliant models, primitive transformations are reduced to a finite number and can be defined using a declarative formalism. The authors of Ref.[11] give fourteen definitions to explain their model transformation approach.

Although so many approaches assume that the transformation rule set can be structured to some extent, the most detailed proposal for the structuring of the rule set comes from the approach brought forward by Biju Appukuttan[12]. It blends the declarative and imperative approach, the declarative part restrain the developer’s choices in a semi-automatic mode, and a more pragmatic method is to give the explicit imperative behavior. It is an overall framework for transformation that allows one to use a variety of different transformation styles. This framework also transparently allows transformations to change style throughout the lifetime of a system. Such transparency is enabled by identification of two distinct subtypes of transformations: relations and mappings, corresponding to declarative and imperative. Transformation is a super type of relation and mapping. In order to make our approach widely accepted and used, we choose this one as our model framework.

3.3 Our model-to-model transformation approach

The OMG Action Semantics specification has already been mentioned as having a mechanism for manipulating instances of UML model elements. The UML Action Semantics extends the UML to be a computationally complete modeling formalism capable of specifying the entirety of the behavior of a system or set of subject domains in a platform-independent form[13]. A UML model which incorporates actions and operations expressed in an action language conformant with the UML action semantics is an executable model[14]. Thus, we can say an action language that conforms to the UML Action Semantics has the expressive power to specify computationally complete, executable models of queries and transformations that operate on MOF models.

In keeping with the tenets of MDA, multiple models, each of which incorporates action language, may be integrated to create an executable platform-independent model of a system. When such an assembly of models executes, the (M0) instances behave in response to stimuli according to the rules and procedures specified in action language. As a metamodel is a model, some metamodels can be integrated to form an executable platform-independent model of a system. So we can say that when an assembly of models that include metamodels executes, the instance, which in this case are M1 instances, behaves according to the rules and procedures expressed in action language. The rules and procedures in that assembly may, as the modeler chooses, specify queries on one (meta)model and the creation of (M1) instance of (M2) classes in another(or the same) metamodel[15].

As for our project, action language may be added as operations on classes in a source model (UML1.X metamodel) so that each class knows how to transform its instance (UML1.X model) into instance (UML2.0 model) of classes in a target model (UML2.0 metamodel), also action language may be added as operations on classes in a target model so that each class knows how to create its instance on the basis of queries invoked on a source model. And in the transform process, we can take the metamodel of the two versions as model (M1) of the whole transformation, and the model of the two versions can take as instances (M0) of the transformation model. So we give out a transform model in Fig.5 which can be used in many kinds of model to model transformation localized not only in UML1.X models but also in UML2.0 models.

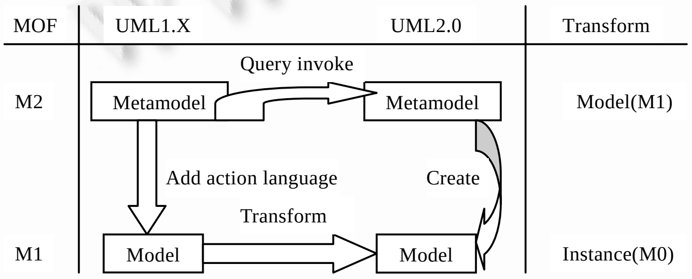

Fig.5 The layer and rules of $\text{UML}_{1.X\text{-}2.0}$ model transformation

Our approach is also kept to the hybrid transformation approach. According to the framework we had given in the previous part, a mapping can refine any number of relations. Because the relations are declarative and the mappings are imperative, we can assume that in our approach, the declarative parts are embedded in the imperative parts, because a set of action language statements implicitly declare instances of classes in the Action Semantics metamodel, such as ASL[15].

3.4 Action semantic language (ASL)

The UML1.4 specification introduces a new Action Semantic language (ASL) which provides a number of low-level constructs that can be composed to provide specifications of actions. The language is compatible with the emerging “Precise Action Semantics” extension to the UML standard. It is an implementation-independent language for specifying processing within the context of an executable UML (xUML) model. The language has been placed in the public domain and may be freely used by modelers and developers. It is in routine use in industry to specify PIM-PSM transformations and transformations to non-OMG languages. The aim of the language is to provide an unambiguous, concise and readable definition of the processing to be carried out by an object-oriented system.

Except ASL, there are many other kinds of action languages, such as BridgePoint Object Action Language, SMALL, TALL, OCL-based, Kabira AS, etc. But only a set of them implicitly declare instance of classes in the

Action Semantics metamodel, and this is the absolutely necessary feature of our framework. And the executable UML models with the UML diagrams being supported by the ASL can allow system developers to build executable domain models and then use these models to produce high quality code for their target systems. The resulting models can be independently executed, debugged, viewed and tested. There are tools that can support this transformation, such as iUML[16]. And ASL has many important features that other action languages do not have. A key feature is that ASL provides facilities formulated at the UML level of abstraction to manipulate the static model. This means that the action is specified in a platform-independent manner. No assumptions are made about middleware, middleware implementation, and implementation language or software design policy. ASL is also used to specify the initial conditions for model execution as well as external test stimuli. In addition, features are provided to allow models to access pre-existing legacy code and other platform specific domains models[17]. So we think ASL is the most suitable action language for describing this transformation.

4 A Case Study of Interaction

4.1 Semantics of interaction

The interaction is the most important metamodel of the interaction package, and it is crucial to the collaboration of others metamodels. An interaction specifies the communication between instances performing a specific task.

UML1.4 Each interaction is defined in the context of collaboration. In the metamodel, an interaction contains a set of messages specifying the communication between a set of instances conforming to the ClassifierRoles of the owning collaboration[3].

UML2.0 An interaction is a unit of behavior that focuses on the observable exchange of information between ConnectableElements[5].

4.2 Abstract syntax

An interaction is a specialization of InteractionFragment (UML2.0 New) and of Behavior. Interactions are now contained within classifiers and not only within collaborations. Lifelines instead of ClassifierRoles model their participants.

In UML1.4, interaction has an association "*context*", specifying the collaboration that defines the context of the interaction. But in UML2.0, it doesn't exist any more, because an interaction do not belongs to collaboration. In order to fit to its new super class and participants, UML2.0 also adds some new association to consummate its feature, "*lifeline*" which specifies the participants in this interaction, and "*fragment*", which specifies the ordered set of fragments in the interaction.

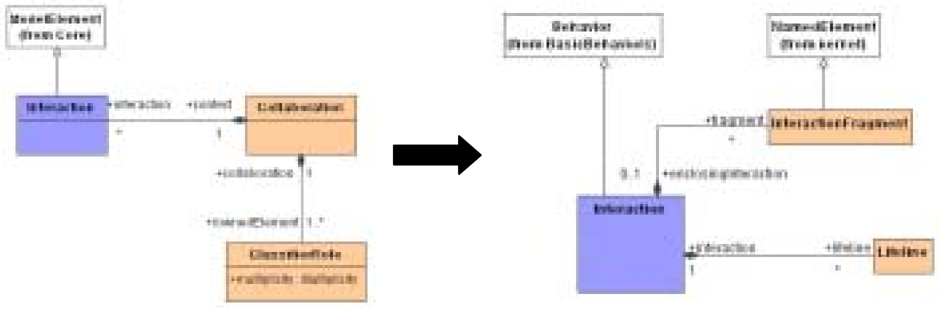

Fig.6 The metamodel of Interaction in UML1.4 and UML2.0

4.3 Transformation rules

Most of our transformation rules are conformable to the rules of ASL. In order to make our approach more efficient, we just change some of them. In ASL, there are three kinds of operations: object scoped, class scoped and domain scoped operation. As for our UML transformation, all elements are metamodels, so we use the class-scoped operation. A class-scoped operation is one that is associated with and provided by a specific class in a domain. For example, this might be used to provide a transformation operation for the class. Such an operation would transform an old class to a new class and set up any initial conditions required. If the class is active, it could then generate a signal to the class to drive it through its lifecycle[18].

Operation Definition Syntax (for class scoped operations):

define function ⟨*operation specification*⟩
  input ⟨*parameter*1 *name*⟩:⟨*parameter*1 *type*⟩,⟨*parameter*2 *name*⟩:...
  output ⟨*result*1 *name* ⟩:⟨*result*1 *type*⟩,⟨*result*2 *name*⟩:...
  ⟨*ASL statements*⟩
enddefine

- ⟨*operation specification*⟩
  Identifies the class scoped operation and conforms to the following syntax:
  ⟨*class key letter*⟩ ⟨*operation number*⟩:⟨*operation name*⟩
  For transformation, we think this part can update to:
  ⟨*core class*⟩ ⟨*classes being changed*⟩:⟨*operation name*⟩

This change can clearly express the relation between the class on changing and the core class of this metamodel, making the mapping between them more legibly.

- ⟨*parameter* 1 *name*⟩:⟨*parameter*1 *type*⟩ ⟨*parameter*2 *name*⟩:⟨*parameter*2 *type*⟩...
  Are comma-separated pairs of formal parameter names and their types that are passed into the operation.
- ⟨*result*1 *name*⟩:⟨*result*1 *type*⟩ ⟨*result*2 *name*⟩:⟨*result*2 *type*⟩...
  Are comma-separated pairs of formal parameter names and their types that are returned by the operation.
- ⟨*ASL Statements*⟩ are the ASL statements that define the method for the respective operation.

The other rules of ASL can be embedded in the operation's ⟨*ASL* Statements⟩. This can efficiently help our transformation. The input and output parts are consistent with the mapping part of the framework we give out above, and the ⟨*classes being changed*⟩ will identify the relation of class being changed to the core class that have been changed.

4.4 Mapping overview

Interaction:	⟨*Interaction*⟩~⟨*Interaction*⟩
Interaction super class:	⟨*ModelElement*⟩~⟨*Behavior*⟩,⟨*InteractionFragment*⟩
Interaction contained within:	⟨*Collaboration*⟩~⟨*Classifier*⟩
Interaction's participants:	⟨*ClassifierRole*⟩~⟨*Lifeline*⟩

4.5 The ASL codes of the transformation

(1) **define** function Interaction: transformation
    input: *Interaction*: *class*
    output: *Interaction*: *class*
  enddefine

This operation is transforming the interaction class, and there isn't any change of this class.

```
(2) define function InteractioSuperClasses: transformation
        input: ModelElement: class
        output: Behavior: class, InteractionFragment: class
          delete ModelElement.unassociation ModeleElement from Interaction
          create Beahvior where Interaction=Behavior.Interaction
          create InteractionFragment where Interaction=InteractionFragment.Interaction
          {InteractionFragmentd}=Interaction-> 〈“fragment”.Interaction〉
    enddefine
```

This operation is transforming the super classes of interaction class. In UML1.4, interaction has a super class ModelElement, but in UML2.0, an interaction is a specialization of InteractionFragment (UML2.0 New) and of Behavior.

```
(3) define function InteractionContainedWithin: transformation
        input: Collaboration: class
             {Interaction}=Collaboration-> 〈“interaction”.Collaboration〉
        output: Collaboration: class, Classifier: class
             create Classifier
             {Interaction}=Classifier-> 〈“interaction”.Classifier〉
    enddefine
```

This operation is transforming the classes within which interaction is contained. In UML1.4, interaction is only contained within collaboration, but in UML2.0, interactions are contained within both the classifiers and collaborations.

```
(4) define function InteractionParticipants: transformation
        input: ClassifierRole: class
             {ClassifierRole}=Collaboration-> 〈“ownedElement”.Collaboration〉
        output: Lifeline: class
          delete ClassifierRole
          create Lifeline
          {Lifeline}=Interaction-> 〈“Lifeline”.Interaction〉
     enddefine
```

This operation is transforming the interaction’s participants. In UML1.4, interaction’s participants are ClassifierRoles, but in UML2.0, ClassifierRole has been replaced by Lifeline.

5 Conclusion and Future Work

In the paper, we compare the UML1.4 and UML2.0 from the top-level of metamodel, also choose a declarative and imperative hybrid framework, and present a UML model transformation method based on Action Semantic. We also describe the transformation of the interaction metamodel with Action Semantic Language compatible with the UML Action Semantics, which shows the feasibility of our approach. The main advantage of our approach is that easy and mathematically precise model transformations can be directly encoded in the standard action specification language of the MDA (and UML) environment, thus providing a smooth integration of formal specifications and industrial standards. In addition, our approach is not limited to this, and also useful in other metamodel or model transformation. The entire encoding is demonstrated on a small example that is still rich enough to cover all the basic rules of our encoding.

Describing the transformations with Action Semantic Language is distinct. All the new classes, associations and attributes can easily be created and transformed. But this method is in progress, and the execution of those codes must be embedded in the state machine of the Executable UML. This will reduce the efficiency of our transformation. So the future work is to solve this question, and it also should improve the veracity of this kind of transformation.

Acknowledgement Thanks to the teachers and classmates in the laboratory for their discussion, cooperation, and contribution, and thanks to specially the office teachers for their helping in my live.

References:

[1] Pender T. UML Bible. Indianapolis: Wiley Publishing, Inc., 2003. 18−42.

[2] Object Management Group. MOF2.0 query/views/transformations (QVT) request for proposals (RFP). OMG Document, 2002.

[3] Object Management Group. OMG unified modeling language specification version 1.4.

[4] Object Management Group. UML infrastructure specification. OMG Document, 2003.

[5] Object Management Group. UML superstructure specification. OMG Document, 2003.

[6] FOLDO. FOLDOC free on-line dictionary of computing. http://wombat.doc.ic.ac.uk/foldoc/

[7] Gardner T, Griffin C, Koehler J, Hauser R. A review of OMG MOF2.0 query/views/transformations submissions and recommendations towards the final standard. OMG Document, 2003.

[8] Bosworth A. Data routing rather than databases: The meaning of the next wave of the web revolution to data management. In: Proc. of the 28th Very Large Data Bases (VLDB). 2002. http://www.cs.ust.hk/vldb2002/VLDB2002-proceedings/

[9] Alcatel, Softeam, Thales, TNI-Valisys, Codeagen Technologies Corp. Response to the MOF2.0 query/views/transformations RFP. OMG Document, 2002.

[10] Bezivin J, Farcet N, Jezequel JM, Langlois B, Pollet D. Reflective model driven engineering. UML, 2003. 175−189.

[11] Song D, He KQ, Liang P, Liu WD. A formal language for model transformation specification. In: Proc. of the Int'l Conf. on Enterprise Information Systems (ICEIS 2005). Springer LNCS, 2005.

[12] Appukuttan B, Clark T. A model driven approach to building implementable model transformations. In:Proc. of the WiSME 2003 workshop of UML 2003. San Francisco, 2003.

[13] Varr´o D, Pataricza A. UML action semantics for model transformation systems. Periodica Polytechnica, 2003,47(3,4):167−186.

[14] Suny_e G, Pennaneac'h F. Using UML action semantics for executable modeling and beyond. In: Proc. of the CAiSE 2001. 2001.

[15] Kennedy Carter Ltd. Initial submission to OMG RFP ad/2002-12-10 MOF query, views and transformations. OMG Document, 2003.

[16] Kennedy Carter Ltd. iUML tutorial for iUML & iUMLite 2.1 NT. Kennedy Carter Ltd, 2000.

[17] Kennedy Carter Ltd. Supporting model driven architecture with eXecutable UML. Ref: CTN 80 V2.2, 2002.

[18] Wilkie L, King A, Clarke M, Weaver C, Raistrick C, Francis P. UML ASL reference guide for ASL language level 2.5 revision D. Kennedy Carter Ltd, 2003.

CHEN Xiu-Hong was born in 1981. She is a master candidate of Wuhan University. Her current research areas are UML, model transformation, software component, etc.

HE Lu-Lu was born in 1978. She is a Ph.D. candidate of Wuhan University. Her current research areas are software component, MDA, etc.

HE Ke-Qing was born in 1947. He is a professor and doctoral supervisor of Wuhan University, and a CCF senior member. His research areas are complex network, software engineering, software interoperation, etc.