Approach of Designing an Executable Meta-Modeling Language in MDA Framework*

WANG Xiaoyan, LIU Shufen, BAO Tie and PENG Jun

(College of Computer Science and Technology, Jilin University, Changchun 130023, China)

Abstract — The aim of MDA is to increase the quality and speed of system development by using modeling techniques. However, the lack of accurate semantic language support has hampered the wide adoption of MDA. This paper presents an executable meta-modeling language xKL design method and a mapping method from xKL to the Java language. metaKernel based on CMOF model is the core of xKL language, which enriches its data types by expanding DataType element, increases expression types by adding ExpType element, improves Operations by adding a variety of basic action information. metaOCL expanded from metaKernel can express constraints between elements. Model mapping tool metaMap provides mapping method from xKL language to Java language and illustrates the mapping rules. The created domain model using xKL language can be directly mapped to the target language using metaMap, so the domain model will never become obsolete if technology changes, and can be developed and reused continuously.

Key words — Meta model, Executable meta-modeling language, Model mapping.

I. Introduction

OMG proposed an open, neutral software development approach called Model driven architecture (MDA)[1] in 2001. The essential aspects of software in MDA are expressed in the form of models, and transformations of these models are considered the core of software development, then models can be transformed into a technical implementation, i.e. a software system. Such an approach can avoid restricting oneself to a specific technology in the early stages of the development process and can ensure a consistent architecture throughout the lifecycle of a software system. However, MDA and its related specifications are still in constant improvement and development, such as MOF[2] and UML[3]. There are a number of problems with them. Firstly, These specifications are short of providing a precise semantics. Whilst their syntax are mostly well specified, the semantics of those syntactic elements are either missing or provided informally with English. Secondly, MOF and UML are not executable—they are designed to be a declarative language. In other words you cannot run a model as defined in the specification, merely define a specification to which any executable program must conform. Thirdly, it’s not clearly defined what is transformation from the most abstract model of a system to the most refined model is, which may include several stages of models. In order to overcome these problems, an executable meta-modeling language is needed which has a precise semantics and can be constructed to meet the different needs and all shortcomings brought about by the imprecise semantics of model-driven software development will be resolved. An executable meta-modeling language enables developers to use powerful language abstractions and development environments that support their development processes. They can create models that are rich enough to permit analysis and simulation of system properties before completely generating the code for the system, and are more reusable and agile. They can manipulate their models and programs in significantly more sophisticated ways than they can code. So software development based on MDA will be further promoted, which can alleviate or solve the problems, such as the complexity, diversity and variability and improve the efficiency of software development.

Meta-modeling is the meta model of a modeling language, the meta-language is used to describe meta-modeling and meta-meta model is the meta-model of meta language. Nowadays, the research on the meta-modeling language or the meta-meta model majors in 2 aspects: One research direction is about meta-meta model constructed by object-oriented methodology, such as MOF and UML class diagram. UML is self-describing model and through a simple extension UML class diagram is approximately equivalent to MOF. Ruth Breu[4] outlines a proposal for the formal foundation of UML that is based on a mathematical system model. A.S. Evans[5] introduced denotational semantics into the UML meta-model, which places emphasis on building a precise core semantics for the UML. MetaGME[6] supports its own metamodelling environment and language based on UML class diagrams with class stereotypes and OCL constraints. EMF[7] conforms to EMOF structure, which implements the EMOF metadata model and metadata operations, but it is not implemented. Hence the result verification and modeling simulation are limited. The other research direction is about graph-based meta-model. The modeling language semantics can be formalized.
with graph transformation by graph-based meta-language, and semantic reasoning and verification can be realized. Hence it can detect the designing error early, but it’s more difficult to achieve. In Moses\cite{8}, type graph is used as meta-language, and attributed graph is used to construct meta-model. Dynamic meta modeling\cite{9} defines the semantics of a Visual modeling language.

The rest of this paper is structured as follows. Section II gives an overview of the construction method of xKL language, Section III describes the methods to extend the various parts of xKL in details, including its grammar and the mapping from xKL to Java, Section IV, taking an network topology as an example, introduces the construction process using xKL, Section V concludes the whole paper and proposes a future research plan.

II. Approach of Constructing an Executable Meta-Modeling Language

Nowadays, developers are sometimes perplexed by the type definitions of UML because these type definitions aren’t suitable for the problem domain of the development system. The idea of designing an executable meta-modeling language which we called xKL in this paper is based on CMOF model. MOF consists of two parts in MOF specification 2.0: EMOF (Essential MOF) and CMOF (Complete MOF), EMOF is the subset of CMOF and its main function is data mapping, but CMOF is used to define other meta-models (for example UML2.0). And neither of them explicitly support executable meta-modeling in a platform independent way. The framework of xKL is indicated in Fig.1.

xKL is composed of 3 parts, MetaOCL, MetaKernel and JavaCC. Construction MetaModel and Behavior MetaModel constitute the MetaKernel. Construction MetaModel conforms to CMOF model with a little modification. xKL enriches its data type elements by expanding DataType element in CMOF, increases expression type elements by adding ExpType element and its sub elements. xKL improves Operations by adding a variety of basic action elements which constitute the behavior MetaModel. After these modifications, a basic executable model which is called metaKernel is constituted. It contains basic static modeling concepts and dynamic modeling concepts. The ultimate goal of xKL is to provide CMOF a precise semantics, which not only makes the model to be based on CMOF standard, but also can be executed automatically, hence the model transformation can be achieved. From the language engineering point of view, xKL is not only a model, but also a language which contains syntax and semantics. This paper uses the JavaCC grammar to define xKL syntax, using the extended OCL language metaOCL to provide constraints among model elements.

III. XKL Language

1. metaKernel

The structure of metaKernel extending from CMOF is as follows:

(1) Data type and expression extension
CMOF model lacks of collection type element and is unable to express a collection which contains some elements with the same feature. metaKernel adds the collection element which is the basic type of object-oriented system through inheritance mechanism and its sub classes, such as Set, Sequence and Bag. Neither of them explicitly support executable meta-modeling in a platform independent way. The framework of xKL is indicated in Fig.1.

xKL is composed of 3 parts, MetaOCL, MetaKernel and JavaCC. Construction MetaModel and Behavior MetaModel constitute the MetaKernel. Construction MetaModel conforms to CMOF model with a little modification. xKL enriches its data type elements by expanding DataType element in CMOF, increases expression type elements by adding ExpType element and its sub elements. xKL improves Operations by adding a variety of basic action elements which constitute the behavior MetaModel. After these modifications, a basic executable model which is called metaKernel is constituted. It contains basic static modeling concepts and dynamic modeling concepts. The ultimate goal of xKL is to provide CMOF a precise semantics, which not only makes the model to be based on CMOF standard, but also can be executed automatically, hence the model transformation can be achieved. From the language engineering point of view, xKL is not only a model, but also a language which contains syntax and semantics. This paper uses the JavaCC grammar to define xKL syntax, using the extended OCL language metaOCL to provide constraints among model elements.

(2) Action extension
It can be seen from the CMOF model that in addition to Operation element, it doesn’t provide any element to describe the behavior of the component. Operation can be instantiated by the object, but CMOF did not give any concepts about action besides Operation, so it may lead to imprecision. The behavior MetaModel expanded from CMOF provides concrete concepts of actions. The Action element in Fig.3 represents the behavior of an element. It has the input, output and the body, and is executed as a method of Operation. There are three parts in expression model: ① PrimitiveAction. It represents the smallest unit of behavior that cannot be broken. NullAction, VarAction, ObjAction and SlotAction are the basic PrimitiveActions. ② CompoundAction. It represents the collection of actions. GroupAction, LoopAction and ConditionalAction are three basic CompoundActions. ③ ExceptionAction. It consists of TryAction and ThrowAction.

2. metaOCL
metaKernel has the computing ability by adding object-oriented concepts to CMOF model, but it can not express constraints between model elements or the query method while OMG OCL language can. However, OMG OCL language is not a programming language, therefore, it is not possible to write program logic or flow control in OCL. And a process or
non-query operation within OCL cannot be invoked because OCL is just a modeling language which is not executable[10]. This paper provides metaOCL language to overcome the shortcomings of OCL language, which extends the metaKernel language and has the ability of OCL language, and the most important is that it can be executed. The metaOCL model is shown in Fig.4. Constraint element is the collection of constraint expressions added to an element, which usually contains some OCLExpressions, such as Iterate, Iterator, LetExp.

3. xKL grammars and semantics

From the xKL extension methods based on CMOF model described above, we know that xKL not only has the characteristics of object-oriented language, but also has the characteristics of a modeling language, so xKL is an object-oriented modeling language. It can use the object-oriented concept to represent model queries, constraints, behaviors and transformation.

In this paper, we use JavaCC grammar to construct xKL text grammar, the construction process is divided into three steps, shown in Fig.5.

(1) To establish the Construction metaModel grammar, which includes basic definitions of types and syntax, and the construction of the basic expression grammar;
(2) To establish the Behavior metaModel grammar;
(3) To establish MetaOCL model constraint language grammar.

The concrete syntax is omitted in this paper.

Translation mechanism is used to realize xKL semantic, it translate xKL into the Java syntax, then it can be run on JVM. So the mapping from xKL to Java needs to be realized, that is to say each element in xKL must map to the Java elements. This paper use metaMap to achieve the mapping from xKL element to Java element. metaMap can record the rela-
relationship between the input models and the output model, it allows more than one input model, but only one output model can be generated. The input models do not depend on the output model. If the input models change, then metaMap must be re-executed and re-generated an output model. A java program composed of a series of packages, a package contains more than one class, a class can contain properties, methods, method names, method bodies, the return value types and parameters. From the construction of xKL language described above, xKL language has the same object-oriented concepts as Java language contains, but the constraint language metaOCL is not the contents of the Java language. So how to realize the mapping from metaOCL elements into Java element is the key problem. Fig.6 is a mapping framework from xKL language to the Java language, the mapping rules are as follows: 

**Rule 1** Package2JavaPackage represents the package mapping from xKL to Java, since Package may include classes, operations, constraints, so Class2JavaClass, Constraints2JavaCheck, Operation2JavaOp need to be called; 

**Rule 2** Class2JavaClass represents the class mapping from xKL to Java, and the class contains the constraints and operations, so Constraints2JavaCheck and Operation2JavaOp are called; 

**Rule 3** Constraints2JavaCheck represents the mapping from constraints contained in xKL class to the check method of Java class. A Constraints may contain many expressions of metaOCL, so OCL2JavaExp is called; 

**Rule 4** OCL2JavaExp represents the mapping from constraint expressions to the corresponding Java expression; 

**Rule 5** Type2JavaType represents the basic data types mapping from xKL to Java. 

The converted xKL language can be run on the Java platform, you can run the model by calling the check method to check whether the model meets the constraints built in the meta model, so you can verify the validity of the model. At the same time, the mapping from xKL to Java language can achieve code generation. Additional method called javaPrint is added to each method which will output the converted language format.

### IV. Example

In the network management, network topology construction is an important part, how to build a right network topology is the first step in building a network management platform. In this paper, we use xKL to build meta model, and establish all kinds of constraints in the meta model. After a model is built, we can run the check method to verify the validation of the model. Finally, a mapping is executed, and some telecom domain components are generated in Java language.

To simulate topology, the first step is to analyze and abstract the network. For a typical IP network, it may contain various types of devices and multiple connections between devices. Network model need to be abstracted according to the specific network. The network topology used in this paper is shown in Fig.7.

The network topology constructed using metaKernel is as follows (see Table 1).

![Fig. 7. The topology of a network](image)

#### Table 1

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Package2JavaPackage represents the package mapping from xKL to Java, since Package may include classes, operations, constraints, so Class2JavaClass, Constraints2JavaCheck, Operation2JavaOp need to be called;</td>
</tr>
<tr>
<td>2</td>
<td>Class2JavaClass represents the class mapping from xKL to Java, and the class contains the constraints and operations, so Constraints2JavaCheck and Operation2JavaOp are called;</td>
</tr>
<tr>
<td>3</td>
<td>Constraints2JavaCheck represents the mapping from constraints contained in xKL class to the check method of Java class. A Constraints may contain many expressions of metaOCL, so OCL2JavaExp is called;</td>
</tr>
<tr>
<td>4</td>
<td>OCL2JavaExp represents the mapping from constraint expressions to the corresponding Java expression;</td>
</tr>
<tr>
<td>5</td>
<td>Type2JavaType represents the basic data types mapping from xKL to Java.</td>
</tr>
</tbody>
</table>

Constraints are added using metaOCL as follows:

1. The port name of d device is unique

```java
context Device
#Constraint PortHaveUniqueName
self.ports->forall(p1| self.ports->forall(p2 | p1.name = p2.name implies p1 = p2))
```

2. The device name can’t be empty

```java
context Device
#Constraint DeviceHasName
self.name!=""
```

A model instance is established using the specific syntax of the meta model described above (see Table 2).

After the establishment of a model instance, you can run the function Check () which will verify the elements in the model instance if there is incompatible with the constraints which is written in meta-model. For example, there is a constraint that the device name should be unique, so if device name is not unique, there will be an error to notify the designer.

The correct model instance can be mapped into the telecommunication components written in Java language. Taking a router component as example, the generated code is illustrated as Table 3.
V. Conclusions and Future Research

MDA is a new software development method than the traditional software development methods, which pays more attention to the role of the model and improves the software development level of abstraction. However, the imprecise semantics of the model language are not executable, hence the transformation from model to code can’t be implemented automatically, so that models in the software development process are meaningless. This paper describes the design method of an executable modeling language xKL and model mapping language metaMap. Validation of models established with xKL language can be checked and Java code can be generated directly with metaMap, so the domain models will never become obsolete if technology changes, and they can continue to be developed and reused.

However, some problems mentioned in this paper, including visual modeling tools designing, the mapping from xKL to C# need to be further studied.

References


WANG Xiaoyan was born in Jilin Province, China, in 1977. She received Ph.D. degree in computer science and technology from Jilin University in 2008. Currently she is a lecturer of College of Computer Science and Technology in Jilin University. Her research interest is in the field of MDA technology, network management technology and cloud computing. (Email: wangxy@jlu.edu.cn)

LIU Shufen was born in 1950. Currently she is a professor in College of Computer Science and Technology of Jilin University. Her research interest is in the area of software programming method based on MDA, network management, work flow technology and computer supported co-operated work. (Email: liusf@jlu.edu.cn)

BAO Tie (corresponding author) was born in Jilin Province, China, in 1978. He received the B.S., M.S. and Ph.D. degrees from Jilin University, Changchun, China, in 2001, 2004 and 2007 respectively all in computer science and technology. Then, he became a lecturer at Jilin University. His research focuses on network management technology, formal description based on Communicating sequential processes (CSP) and model-driven approach. (Email: apche@126.com)

PENG Jun was born in Chongqing, China in 1981. He received B.S. degree and M.S. degree from Jilin University in 2004 and 2006 respectively, both in computer science and technology. Currently he is a doctor student in Jilin University. His research interest is in the field of MDA technology, software architecture and component technology. (Email: pengjun@email.jlu.edu.cn)