基于本体语义的模型映射研究*)

侯金奎[1] 王 锋[2] 张 睿[2]

(山东大学计算机科学与技术学院 济南 250061)[1] (山东省电子产品监督检验所 济南 250014)[2]

摘 要 模型转换是模型驱动开发的一项关键技术,模型间的映射关系是模型转换的基础和依据。通过对模型描述语言的语法结构和语义表达特性进行抽象分析,基于本体语义映射,对不同层次的模型映射进行了分类并加以形式化定义,进一步探讨了不同抽象层次模型之间映射关系的建立过程以及所应遵循的基本原则。以 UML 类模型到 C 语言模型的映射为例进行了阐述。这不仅可为模型转换的具体实现提供理论指导,还为验证模型之间映射规则的可行性和正确性提供依据。

关键词 模型驱动开发,模型映射,本体语义,语义一致

Research on Model Mapping Using Ontology Semantics

HOU Jin-kui[1] WANG Feng[2] ZHANG Rui[2]

(School of Computer Science and Technology, Shandong University, Jinan 250061, China)[1]

(Shandong Electronic Products Supervision and Inspection Institute, Jinan 250014, China)[2]

Abstract Model transformation is a key technology of model-driven software development, while the mapping relations between different models are the foundation and basis for the transformation. Based on ontology semantic mapping, a classification for different level model mappings was proposed and defined formally by abstractly analyzing the characteristics of syntax and semantic features of modeling languages. On this basis, a further study was conducted to explore the definition process for mapping relations and the cardinal principles should be followed. The UML-based class model used as a source and the C programming language used as the target are shown in the case study to help interpreting the ideas. It may not only be a theoretical guidance for model transformation, but also can be a measurement for validating the mapping rules between models at different abstract levels.

Keywords Model-driven development, Model mapping, Ontology semantics, Semantic consistency

1 引言

模型驱动(MDD, Model-driven Development)的软件开发是软件工程技术的研究热点和发展趋势,它通过提升抽象层次来应对软件开发的复杂性[1]。模型从最初作为黑盒的系统规范开始,经过一系列严格的转换过程(每一步转换都添加了层次细节),最终实现为系统构件[2]。已经提出的模型转换方法[3,4]大都侧重于不同抽象层次模型之间具体实现技术的讨论,而在映射关系的定义原则,以及映射关系的正确性验证等方面,还缺乏坚实的理论基础[5],有待于更深入、形式化的研究,以支持错综复杂的模型转换。

由于不同的模型描述方法对问题进行分析和设计的关注点以及语义表达方式等方面存在差别,使得不同抽象层次的模型在组织结构和信息表达方式上往往存在较大的差异,这种差异使得模型之间的映射关系错综复杂,难于直接定义。文献[6,7]对模型描述语言的语法结构和语义表达做了详细的介绍,但没有涉及模型转换过程中语义的迁移和重构。文献[8]从元模型语义的角度分别介绍了基于 UML 和 EDOC 的平台无关模型到 JAVA、Web service 和 JWSDP 三种目标平台模型的映射关系,并简单讨论了模型描述之间的异构性对映射关系定义的影响,但没有给出一般情况下的解决方案。文献[5]以扩展形式化语言的方式对不同抽象层次模型之间的转换分类做了说明,该方法可以在一定程度上降低模型描述之间的异构性,但是不能完全消除,模型之间的映射仍难以直接定义。自然语言机器翻译研究的理论表明,实现不同语言之间正确转换的前提是源语言和目标语言具有相同或相似的语义表达特性[9],模型驱动开发中不同抽象层次模型间的转换也是如此。

本文通过对模型描述语言的语法结构和语义特性进行抽象分析,基于本体语义映射,在对不同层次的映射进行了分类并加以形式化定义的基础上,探讨了模型之间映射关系的建立过程以及所应遵循的基本原则。本文结构如下:第 2 节在讨论模型描述之间异构性的基础上,介绍了模型转换中的语义一致性要求;第 3 节对不同层次的映射进行了分类并加以形式化定义;第 4 节以 UML 类模型和 C 语言分别作为源和目标,说明了不同抽象层次模型之间映射关系的建立过程;最后对全文进行总结,并指出进一步研究的方向。

2 模型和模型映射

模型是软件系统的抽象,是用一种给定的形式化语言对系统的功能、结构和行为进行的描述[10]。模型描述之间的差异表现为三个层次:语法、结构和语义。语法级的异构是指数据类型、格式的差异;结构级的异构是指数据结构、接口和模式上的不同;语义级的异构则是指不同领域专用词汇含义上的差异。如用 UML 描述的平台无关模型有多种图元对象,而基于某一特定平台的目标模型一般具有特定的应用模式,其数据内容和组织结构相差很大。解决模型描述异构问题的根本途径是构建有效的语义映射机制,确保最终得到意义上等价的系统表达[6,9]。这种映射机制应是形式化的,既便于人的理解,也便于机器处理。

*)山东省科技发展计划项目(2006GG2201009)。**侯金奎** 博士研究生,主要研究方向为模型驱动开发、软件体系结构和形式化方法;**王 锋** 研究员,主要研究方向为网络安全、语义网络;**张 睿** 工程师,主要研究方向为信息安全、语义分析。

原则上来说，各种成熟的模型描述语言（或程序设计语言）在某种技术意义上是等价的[11]。如果有用的特征在一种语言里没有直接提供，一般可以通过深入思考用某种方式模拟来实现。例如，在缺乏抽象机制的语言里，注释和命名规则可以用于模拟模块结构，迭代器可以通过函数和静态变量模拟。在缺少递归的语言里，可以从递归的伪代码出发，通过机械式的手工变换，推导出一个迭代程序。在缺少命名常量和枚举类型的语言里，采用初始化一次后决不修改的变量，可以使代码更容易阅读，也更容易维护。语义是信息表达的内在涵义，其意义与上下文环境有关。参考文献[6,7,12]，本文作如下定义：

定义 1（语义一致性） 指将两个不同的元素集合 U 和 V，分别作为某个应用系统 APP 的输入项，APP 运行后，得到两个输出结果 APP(U) 与 APP(V)，它们在意义表达和功能操作上完全等价，或是大致等同，记作 APP(U)≌APP(V)。

定义 2（语义一致的模型映射） 就是在不同模型描述语言的建模元素之间建立的一种映射关系 MAP，用它作用于两个异类的概念模式 X 与 Y 后，会得到语义对等的两组模型元素集合的表达，即：MAP(X)⇔MAP(Y)。

一致性条件可以用语义领域模型上的语法来定义。一般情况下，语义一致性分为水平一致性和垂直一致性两种：水平一致性存在于同一抽象层次、对同一系统从不同的角度进行描述的模型之间，这些模型可以使用不同的语言来描述，但不包含互相矛盾的概念。如 UML 状态图和时序图，从不同的角度描述了系统的同一个过程。垂直一致性存在于同一系统、不同抽象层次的模型描述之间。比如要精化一个模型，就要保证精化后的模型与精化前的抽象模型描述之间没有矛盾（如状态图 S 和其精化后的版本 S'，要求 S 中所有的可达状态在 S' 中也是可达的）。

模型转换是依据变换定义由源模型自动产生目标模型的过程[10]。从操作的观点看，模型转换是一个可停机算法，它对一个（或一组）模型进行结构的改变和语义的迁移；从功能的观点看，模型转换是一个功能转换，它将一组模型从一个或多个领域映射到相同（或者不同）领域中的另一组模型。模型映射规则是模型转换的基础和依据，包含了由一种模型向另一种模型转化的规约说明[10]。支持 MDD 的模型转换中，需要清楚地理解系统的建立意图、不同层次模型的概念和符号集以及概念结构之间的关系，使用专用的语义描述机制对源和目标模型（或目标代码）中的动态与静态机制分别加以描述，然后建立两者的语义模型，利用语义对等的原则，构造出源模型和目标模型建模元素之间的对应关系，从而进一步在两种不同形式的语义项之间建立语义映射机制，最终实现不同层次模型之间的转换。

3 模型映射的层次

基于本体语义映射[13]，不同抽象层次模型之间映射关系的定义大致可以分为类型之间的映射和结构之间的映射两个层次。

3.1 类型映射

从构造的观点看，类型可分为基本类型和复合类型[10]，前者如整数、字符、布尔、实数等，后者是通过对一个或几个基本类型应用某个类型构造符（如 record、array、set 等）而创建的。

定义 3（类型映射） 指把一种模型描述语言中定义的类型转换为另一种模型描述语言中定义的类型，这两种类型语义一致。

由定义 1 和定义 2，按照语义对应关系，不同模型描述语言间的类型映射可以划分为一致（conform）映射和非一致（non-conform）映射。一致的类型映射又可分为两种：简单映射和扩展映射。简单映射是指在目标模型语言中直接存在和源模型语言对应的类型，并且其语义一致。扩展映射是指在目标建模语言中不存在和源语言直接对应的语言类型，但可以通过特定的组合方式来模拟出这种类型。非一致映射是指目标语言中没有和源语言对应的数据类型，并且也无法通过模拟给出与其语义等价的类型定义。

3.2 结构映射

定义 4（结构映射） 指把在源模型中定义的模块转换为用目标建模语言定义的与其语义等价的模块。

最简单的结构映射是一对一的映射，称之为直接映射，把一个结构映射为另一个结构。例如 UML[14] 定义中的类结构可以直接映射为 Java 语言中的 Class。由于不同模型描述机制的差异，结构映射相当复杂。一般来说，结构映射可分为直接映射、分解映射和组合映射等，如图 1 所示。

在实际的模型中，结构间的映射关系如果为多对多（[N: M]），则可通过添加中间结构定义的方式简化成多对一（[N: 1]）和一对多（[1: M]）的关系。基于此，本文只对上述 3 种映射作如下定义：

定义 5（直接映射） 结构间的直接映射可以表示为一个四元组 $M=(S_A, D_A, R, v)$，其中 S_A 表示一个源模块结构，D_A 表示目标模块结构，R 表示映射关系（用 $S_A \vdash_R D_A$ 来表示）。v 表示映射的信任度，其取值范围为(0,1)，其取值主要取决于模块概念之间的语义相似度和上下文环境，数值越大代表映射的信任度越高。

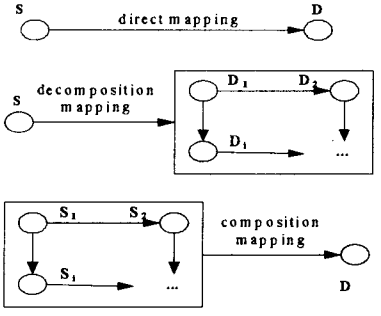

图 1 结构映射的基本形态

定义 6（分解映射） 分解映射可以表示为一个六元组的 $D_m=(S_A, F, D_A, A_{CD}, R_D, v_D)$，其中：

① S_A 表示一个源模块结构，D_A 表示由若干子模块 D_i（$i \leqslant n$）组合而成的目标模块结构；

② F 是一个由直接映射表达式构成的集合，$F=\{F_i \mid F_i = S_A \vdash_R D_i,\ 1 \leqslant i \leqslant n\}$，$F_i$ 代表源结构 S_A 到目标组合结构 D_A 中的一个子结构 D_i 的直接映射；

③ A_{CD} 表示目标组合 D_A 中各子结构之间的包容依赖关系[21,22]构成的集合，可以表示为 $\{D_i \subseteq_k^d D_j \mid 1 \leqslant i, j \leqslant n, 1 \leqslant k \leqslant n(n-1)/2\}$，其中序号 i，j，k 表明了子结构 D_i 和 D_j 及包容依赖关系 $\subseteq_k^d$ 在整个组合中的顺序；

④ R_D 表示源结构 S_A 到整个目标组合结构 D_A 的映射关系，即 $S_A \vdash_{R_D} D_A$；

⑤ v_D 表示这个分解映射的信任度。

定义 7（组合映射） 组合映射可以表示为一个六元组的 $C_m=(S_A, A_{CS}, D_A, L, R_C, v_C)$，其中：

① S_A 表示一个由若干子模块 S_i（$i \leqslant n$）组合而成的源模

块结构，D_A表示目标模块结构；

② A_{CS}表示源组合结构 S_A 中各子结构之间的包容依赖关系构成的集合，可以表示为 $\{S_i \subseteq_k^d S_j \mid 1 \leqslant i, j \leqslant n, 1 \leqslant k \leqslant n(n-1)/2\}$，其中序号 i，j，k 表明了各个子结构 S_i 和 S_j 及包容依赖关系 $\subseteq_k^d$ 在整个组合中的顺序；

③ L 是一个由直接映射表达式构成的集合，$L=\{L_i \mid L_i = S_i \vdash_R D_A, 1 \leqslant i \leqslant n\}$，$L_i$ 代表源组合结构 S_A 中的一个子结构 S_i 到目标结构 D_A 的直接映射；

④ R_C 代表整个源组合结构 S_A 到目标结构 D_A 的映射关系，即 $S_A \vdash_{R_C} D_A$；

⑤ v_C 表示这个组合映射的信任度。

3.3 结构融合

模型间的映射不仅要进行源模型到目标模型的类型映射和结构映射，同时要求目标模型的各组成模块要保持它们在源模型中的语义特性和约束关系。为此，我们作如下定义：

定义 8(结构融合) 假定 $A_i(1 \leqslant i \leqslant n)$ 为 n 个不同的结构，I 为一个有限的约束关系集合，当 A 和 A_i 满足下列条件时，结构 A 被称为 $A_i(1 \leqslant i \leqslant n)$ 的融合：

①存在 n 个映射 $M_i(1 \leqslant i \leqslant n)$ 分别表示从 $A_i(1 \leqslant i \leqslant n)$ 到 A 的映射；

②$(\forall x, y \in A_i)\ x \prec y \Rightarrow M_i(x) \prec M_i(y)$；其中 $1 \leqslant i \leqslant n$，$\prec$ 表示一种语义层次关系；

③$(\forall x, y)\ (x \in A_i \wedge y \in A_j) \wedge (x\ \mathrm{op}\ y) \in I \Rightarrow M_i(x)\ \mathrm{op}\ M_j(y) \in I$；op 表示一种约束关系。

条件②的含义是在融合结构中各子结构的语义层次仍然得到保留；I 可以表示为被融合的各子结构之间的互操作性。条件③的含义是在融合结构中，各子结构之间的约束关系仍然得到保留。

由以上定义，模型之间映射的集成可以分为 3 个步骤，分别描述如下：

第一步称为直接映射的结构融合，这一个步骤主要是根据已经存在的直接映射关系(包括类型映射)构建一个融合连接列表，这里的直接映射也包含复杂映射中的直接映射关系。

第二步称为复杂映射的结构融合，主要对复杂映射中不存在直接映射的各个结构之间建立映射关系，包括分解映射和组合映射两种情况。

第三步称为系统结构融合，依据各模块的语义特性和模块间的约束关系，最终把上述两步得到的多个子目标模块融合为一个整体的目标结构模型，从而得到源模型到目标模型的全部映射关系。

4 实例研究

本节以 UML 类模型到 C 语言模型的映射为例来阐述模型映射关系定义的过程和有关的性质。

4.1 UML 类模型语义特征

在基于 UML 的类模型中[14]，表达语义特征的模型元素主要包括：类(class)、属性(attribute)、方法(operation)、继承(Inheritance)、聚合(aggregation)、关联(association)等。继承(Inheritance)关系表达了建模概念间的子类/超类关系，描述的是一种层次化结构。关联关系的多重(cardinality)特性描述了一个类对象作为另一个类对象的属性时的数量关系。语义模型的这些基本特征通过不同层次的组合形成复杂的模型描述。

模式映射是发生在两个模式之间的一种等价刻画关系。UML 类模式可用如下的 4 元组表示：$\langle C, L_A, L_I, F\rangle$。其中：$C$ 为类、属性、方法等模型元素的集合；L_A 是关联关系标号的集合；L_I 是泛化/特化关系标号的集合，规定

① $A \subseteq C \times L_A \times C$ 为表示关联关系的集合；

② $I \subseteq C \times L_I \times C$ 为表示继承关系的集合；

③ $F: A \rightarrow \{[0:1], [1:1], [1:N], [N:1], [N:M]\}$ 为一个将级联级数与一个关联关系联系起来的函数。

面向对象中的继承关系包括多重继承、互斥继承、与或继承等[14]，本文只简单考虑单重互斥继承的情况。

下面讨论类模型的性质，用小写字母 $a,b,\cdots$ 表示属性边和方法边；小写字母 $p,q,r,\cdots$ 表示类(或类型)；大写字母 $U, V,\cdots$ 表示继承边。先定义如下符号：

1)由 $A \subseteq C \times L_A \times C$，如果 $(p, a, q) \in A$，那么记作 $p \xrightarrow{a} q$，意思是类 p 的任何实例都有一个 a 属性，类型是 q；

2) 由 $I \subseteq C \times L_I \times C$，如果 $(p, U, q) \in I$，那么记作 $p \overset{U}{\Rightarrow} q$，表示 p 是 q 的特化，p 的任何实例也是 q 的实例；

3) 对于 $p \overset{U_1}{\Rightarrow} r_1 \overset{U_2}{\Rightarrow} r_2 \cdots \overset{U_m}{\Rightarrow} r_m \cdots \Rightarrow q$，或者 $p=q$，记作 $p \overset{*}{\Rightarrow} q$。

则类模型有如下基本性质：

①如果 $p \xrightarrow{a} q_1$，$p \xrightarrow{a} q_2$，那么 $q_1 = q_2$；

②如果 $p \Rightarrow q$，$q \xrightarrow{a} r$，那么 $p \xrightarrow{a} r$；

③如果 $p \xrightarrow{a} s$，$s \Rightarrow r$，那么 $p \xrightarrow{a} r$。

式①说明类之间的关系，除了参与联系的类外，属性的名称也参与类的表示。

式②说明了特化关系对属性的继承性，即如果类 p 是类 q 的特化，类 q 有 a 表示的属性 r，则类 p 也有用 a 表示的属性 r。

式③从另外一个角度说明了属性对特化关系的保持，即如果 p 有 a 表示的属性 s，而 s 是 r 的特化，则 p 有用 a 表示的属性 r。

由定义 8，我们知道 UML 类模型的映射不仅要进行类型映射和结构映射，同时还要保持上述语义特性和约束关系。

4.2 UML 类模型到 C 语言模型之间的映射

类是对特定数据的特定操作的集合体，它包含数据和操作两个范畴。在 C 语言中没有直接与之对应的类型或结构，但是我们可以通过结构(struct)和函数来模拟类的实现。以下描述是对几种面向对象特性进行模拟：

(1)使用 struct 组合函数指针模拟纯虚类。

```
typedef struct {
void ( * Fool)();
char ( * Foo2)();
char * ( * Foo3)(char * st);
}MyVirtualInterface;
```

(2)函数名相同，利用标志位调用子函数，可模拟函数重载。

```
Long Nilan(int FunctionCode, Parameter * ){
Switch (FunctionCode) {
   Case SendPacket: send(…)
   Case ReceivePacket: receive(…)
  ……
}
}
```

(3)封装

封装是为了保护对象内部的实现细节，UML 中有 private/package/public 等可见性选项，以适应于不同的情况。在 C 语言中可通过如下方式实现：

成员函数的隐藏：在 C 文件中定义函数时，前面加 static 关键字，并且不要把函数放在头文件中，同时将每个类放在独立的文件中。这样可以把函数的作用范围限于当前文件内，当前文件只有类本身的实现，即只有当前的类自己才能看到这些函数，从而达到隐藏的目的。

数据成员的隐藏：把私有数据信息放在一个不透明的 private 变量中，只有类的实现代码才知道它的真正定义。如：

```
struct _LrcBuilder{
LrcBuilderBegin on_begin;
LrcBuilderOnIDTag on_id_tag;
char private[1];
}
```

(4)继承

由于结构(struct)在内存中的布局与结构的声明顺序一致,所以在C语言中可用结构(struct)来模拟实现类的继承。

(5)多态(polymorphism)

在C语言中可以用函数指针来实现多态,函数指针定义了函数的原型,即它的参数和返回值的描述以及函数的意义,不同的函数可以有相同的函数原型。在不同的情况下,让函数指针指到不同的函数实现上,就实现了多态。

(6)类之间的关联关系

在表示类A的结构(struct)中定义一个指向类B结构的指针变量,以此来表示它们之间的关联关系。如果关系的重数大于1,则可以用指针数组或链表进行表示。

面向对象的许多特性都可以用C语言进行模拟,如delegates、forwarding messages、dynamic type checking等等,限于篇幅,不能一一列出。下面用一个简单例子来说明UML类模型到C语言模型映射的实现。当然,C语言模拟面向对象实现的方法有多种,我们在此采用一种较简单的方案。UML类模型片断如图2左半部分所示,其对应的C语言模型片断如右半部分所示。从图2可以看出,它们之间的映射关系包括直接映射、分解映射和组合映射,最终经过结构融合后的C语言代码实现如下:

```
#ifndef C_Class
#define C_Class struct
#endif
C_Class A {
  C_Class A *A_this;
  void (*Foo)(C_Class A *A_this);
  int a;
  int b;
C_Class C{
…;
}
C_Class B{ // B inherit from A
  C_Class B *B_this;
  void (*Foo)(C_Class B *B_this);
  int a;
  int b;
  int c;
  C_Class C *cObject;
}
void B_F2(C_Class B *B_this)
{
  printf("It is B_Fun\n");
}
void A_Foo(C_Class A *A_this)
{
  printf("It is A. a=%d\n", A_this->a);
}
void B_Foo(C_Class B *B_this)
{
  printf("It is B. c=%d\n", B_this->c);
}
void A_Creat(C_Class A * p) // constructing functions
{
  p->Foo=A_Foo;
  p->a=1;
  p->b=2;
  p->A_this=p;
  }
void B_Creat(C_Class B* p) // constructing functions
{
  p->Foo=B_Foo;
  p->a=11;
  p->b=12;
  p->c=13;
  p->B_this=p;
}
… …
int main(int argc, char* argv[])
{
  C_Class A *ma, a;
  C_Class B *mb, b;
  A_Creat(&a); // instantiation
  B_Creat(&b); // instantiation
  mb=&b;
  ma=&a;
  ma=(C_Class A *)mb; // pointers for polymorphism
  printf("%d\n",ma->a);
  ma->Foo(ma); // polymorphism
  a.Foo(&a); // not polymorphism
  B_F2(&b); //call to member functions
  … …
  return 0;
}
```

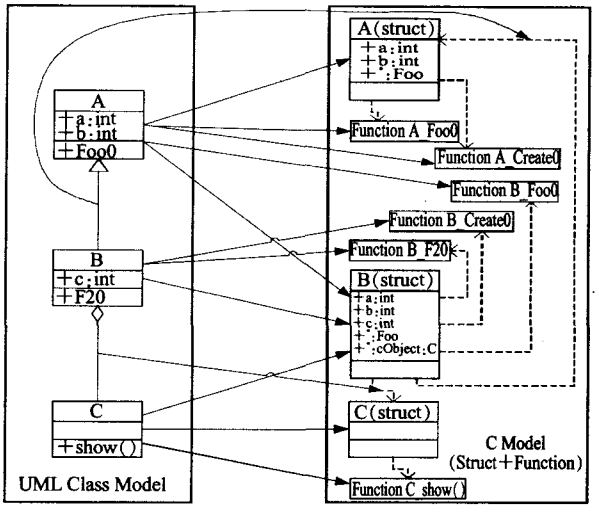

图2 UML类模型到C语言模型的映射

结束语 本文从模型描述语言的抽象分析出发,基于本体语义映射,对模型间不同层次的映射进行了分类,探讨了模型间映射关系的建立过程以及所应遵循的基本原则,可为不同抽象层次模型之间映射关系的正确性验证提供依据,从而对模型驱动的软件开发提供有力的支持。

今后的工作包括:(1)进一步完善类型映射和结构映射的形式化描述,增强其语义表述能力;(2)进一步探讨模型映射所应保持的语义特性,强化不同抽象层次模型间的一致性检验。

参考文献

[1] Hailpern B, Tarr P. Model-driven development: The good, the bad, and the ugly. IBM Systems Journal, 2006,45(3):451-461

[2] Balmelli l, Brown D, Cantor M, et al. Model-driven systems development, IBM Systems Journal, 2006,45(3):569-585

[3] Gardner T, Griffin C, Koehler J, et al. A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard, OMG Document. http://www.omg.com/docs/ad/03-08-02.pdf, 2003

[4] Czarnecki K, Helsen S. Feature-based survey of model transformation approaches. IBM Systems Journal, 2006,45(3):621-644

[5] Caplat G, Sourrouille J L. Model Mapping Using Formalism Extensions. IEEE Software, 2005,22(2):44-51

[6] Harel D, Rumpe B. Modeling Languages: Syntax, Semantics and All That Stuff. Technical paper number MCS00-16. The Weizmann Institute of Science. http://www.cs.york.ac.uk/puml/, 2000

[7] Harel D, Rumpe B. Meaningful modeling: what's the semantics of "semantics"?. IEEE Computer Magazine, 2004,37(10): 64-72

[8] Bezivin J, Hammoudi S, Lopes D, et al. Applying MDA approach for Web service platform//Proceedings of Enterprise Distributed Object Computing Conference, 2004. Monterey, California, USA, 2004:58-70

[9] Dahl V, Saint D P. Natural Language Understanding and Logic Programming. London:Elsevier Science Publishers, 1985

[10] Kleppe A, Warmer J, Bast W. MDA Explained. The Model Driven Architecture: Practice and Promise, Boston: Addison-Wesley, 2003

[11] Scott M L. Programming Language Pragmatics. San Francisco: Morgan Kaufmann, 2000

[12] Stefik M J. Introduction to Knowledge Systems. San Francisco, California: Morgan Kaufmann, 1995

[13] Kwon J, Jeong D, Lee L-S, et al. Intelligent Semantic Concept Mapping for Semantic Query Rewriting/ optimization in Ontology-based Information Integration System. International Journal of Software Engineering and Knowledge Engineering, 2004,14(5):519-542

[14] Object Manage Group. Unified Modeling Language Specification. http://doc.omg.org/formal/03-03-01, 2003