Int J Software Informatics, Vol.X, No.X, February 2011, pp. xxx-xxx E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
 Tel: +86-10-62661040

Unifying the Semantics of Models and Meta-Models in the Multi-Layered UML Meta-Modelling Hierarchy

Lijun Shan[(1)] and Hong Zhu[(2)]

(1) National Digital Switching System Engineering and Technological Research Center,

Zhengzhou, China, Email: slj@ndsc.com.cn

(2) Department of Computing and Electronics, Oxford Brookes University,

Oxford OX33 1HX, UK, Email: hzhu@brookes.ac.uk

Abstract UML is defined through metamodelling in a four-layer metamodel hierarchy, where metamodels and meta-metamodels are also presented in the form of UML class diagrams. However, the meanings of models and metamodels as well as the basic concepts involved in modelling and metamodelling are not precisely defined in the OMG documentations. In the past few years, a large amount of research efforts on the formalisation of UML semantics has been reported in the literature, but how to formalise the metamodel hierarchy still remains an open problem. This paper presents a framework of unified formal semantics of the metamodel hierarchy. It is based on our previous work on the formal semantics of UML, in which we proposed the notions of descriptive semantics and functional semantics as two separate aspects of UML semantics. The former describes the structure of a model's instances, and the latter characterises the functional and behavioural properties of its instances. This paper further develops this approach by generalising it to metamodels and meta-metamodels. We prove that the semantics of models, metamodels and meta-metamodels can be defined in a unified way. The basic concepts involved in the metamodel hierarchy, such as subject domain and instance-of relation, can also be precisely defined based on the unified semantics framework.

Keywords: Software models, Unified Modelling Language UML, Metamodel, Metamodel hierarchy, Formal semantics.

Authors (Shan LJ, Zhu H). Title (Unifying the semantics of models and meta-models in the multi-layered UML metamodel hierarchy). ***Int J Software Informatics*****, 201x, x(x): xxx-xxx. http://www.ijsi.org/1673-7288/4/i55.htm**

1. Introduction

Models are created and used as the main artefacts of software engineering in the model-driven development methodology. By raising the level of abstraction in software development, model-driven engineering (MDE) facilitates a wide range of automation from architectural design to integration, testing, maintenance and evolution. With the introduction of the Unified Modelling Language (UML), MDE has become very popular today with a large body of practitioners and a wide availability of supporting tools. However, the lack of a rigorous definition of the semantics of UML has been a long lasting issue.

1.1. UML and Its Metamodel Hierarchy

UML is defined through metamodelling; i.e. a metamodel is employed to specify the UML modelling language. A metamodel is a model of some syntactically valid models. Due to the need to define the syntax and semantics of the metamodel, a meta-metamodel is further specified. This leads a four-layer metamodel hierarchy, where a model at layer i is an instance of some model at layer (i+1), for $i \in \{0, 1, 2\}$. Following the terminology used in the UML documentation [2], in the sequel we write 'a M_i model' to denote 'a model at layer i'. In particular, a system in the real world is regarded as an M_0 model, which is an instance of a user model (an M_1 model) in the UML language. The metamodel of UML is an M_2 model. The meta-metamodel of UML, called MOF (MetaObject Facility) model, is the only M_3 model in the four-layer metamodel hierarchy. MOF is intended to be the core of many MDE technologies including UML, CWM (Common Warehouse Metamodel), SPEM (Software & Systems Process Engineering Metamodel), XMI (XML Metadata Interchange), etc.[3]. According to the UML and MOF documentations [2, 3], the hierarchy is allowed to have more than 4 layers.

The metamodel and the meta-metamodel of UML are actually defined in the UML's class diagram notation. Therefore, this metamodelling approach is reflective in the sense that the modeling language is defined in its own notation. Because the notation of UML class diagram is fairly self-descriptive, this approach works well to some extent. UML class diagram incarnates the idea of object-orientation using nodes to denote classifications of objects and edges to denote relationships between objects. In fact, a metamodel can be regarded as a representation of the ontology underlying a modelling language. In the metamodel of UML, for instance, concepts such as Class, Property and Generalisation are represented as classes and depicted as nodes in a class diagram, and generalisation/specialisation and whole-part relationships between the concepts are represented as inheritances and compositions and depicted as edges between the class nodes. In the same way, concepts used in a metamodel can be classified and depicted in a class diagram at a higher layer, i.e. a meta-metamodel.

1.2. Problem Identification

However, this appealing feature of reflective uses of class diagrams in modelling and meta-modelling imposes a great challenge to defining the semantics of UML. That is, can the semantics of UML class diagram be applied to all layers uniformly in the metamodel hierarchy?

Although the metamodel hierarchy is fairly well described and intuitively understandable, the basic notions involved in modelling and metamodelling are not precisely and rigorously defined in the OMG documentations. A key question we are concerned with is the exact

meaning of the 'instance of' relation between a M_i model and a M_{i+1} model in the metamodel hierarchy. According to the UML documentations, real world systems or software systems can be regarded as instances of a UML model. However, little has been said about how to judge whether a system is an instance of a model. Take a simple class diagram that contains one and only one class node labelled with identifier *A* as an example. It can be interpreted in any of the following ways, while the official UML documentation does not specify which one is correct.

- *There is **only one** class in the system and it is **named A**.*
- *There is **at least one** class **named A** in the system (which may have other classes).*
- *There is **only one** class in the system and **its name does not matter**.*
- *There is **at least one** class in the system and **its name does not matter**.*

In our previous work [1, 4], we argued that each of the above interpretations of the instance-of relation between real world systems and UML models has its own role in software development. Therefore, all of them should be regarded as valid semantics of UML models. In order for the semantics of UML to incorporate all these interpretations, we have introduced the notion of *usage context* of models. Given a specific usage context, the hypothesis on how to interpret a model can be explicitly described as a part of the semantics of the model. However, the instance-of relation between models and metamodels cannot be so flexible. For example, given a metamodel which contains only one class node named *Classifier*, it can only be interpreted to: there is one and only one type of elements in the model, and the type is Classifier. A model that contains elements of other types is not an instance of the metamodel, because such types are undefined. The above two examples reveal that the instance-of relation between M_0 and M_1 is different from that between M_1 and M_2. A question is: can we identify and formally specify the usage contexts of class diagrams for their uses as metamodels and meta-metamodels?

Considering the whole multi-layer metamodel hierarchy, the above questions can be generalised into: (a) What are the relationships between any two models at adjacent layers in the metamodel hierarchy? (b) Can the semantics of models at different layers be unified in one rigorous and precise semantic definition?

This paper addresses these problems with a unified semantic framework for the metamodel hierarchy. In our previous work on the formal semantics of UML, we have proposed the notions of descriptive semantics and functional semantics as two separate aspects of UML semantics. The former describes the structure of a model's instances by specifying element types that can be used in the models' instances and relationships between the elements, while the latter characterises the functional and behavioural properties of its instances. This paper further develops this approach by generalising it to a unified definition of the semantics of metamodels and meta-metamodels. In fact, the framework can be extended to any number of layers of metamodelling. Descriptive semantics and functional semantics of models are defined through two mappings from class diagrams to predicate logic formulas, respectively. The functional semantics of a model at any layer can be specified independent of its descriptive semantics and then integrated with descriptive semantics to form a complete semantics of the model. The basic concepts involved in the metamodel hierarchy, such as instance-of relation and subject domain, are precisely defined. The valid instances of a M_i model M are, mathematically speaking, structures in the signature determined by the model M and satisfying the formulas that represent the descriptive and functional semantics of M. The subject domain of a modelling language, i.e. the collection of systems that can be described by the language, can then be defined as the set of instances of the metamodel specifying the language.

1.3. Organisation of the Paper

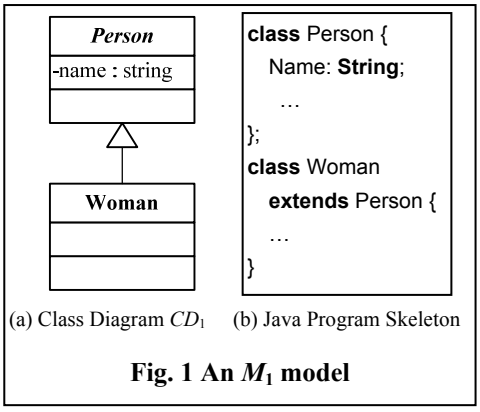
The paper is organised as follows. Section 2 outlines our approach. Section 3 formally defines the basic concepts of the UML metamodel hierarchy. Section 4 is devoted to the descriptive semantics of models at all layers of the hierarchy. We present a set of rules that translate a model into a set of descriptive statements. We also identify the context of using class diagrams as M_i layer models for $i>1$, and specify the context as a set of rules that derive formulas from models. Section 5 presents a set of axioms of OO concepts as the static functional semantics of UML models. Section 6 integrates the descriptive semantics and functional semantics by a set of rules that derives a set of statements representing the functional semantics of models. Section 7 discusses the application of the formal semantics of UML in model-driven software development. Section 8 compares our work with related work. Finally, Section 9 summarises the main contributions of this paper and discusses future work.

2. Overview of the Proposed Approach

As Seidewitz pointed out [5], a software model, like models in any other scientific disciplines, is '*a set of statements about some system under study*', where statements are expressions that can be evaluated to a truth value with respect to the modelled systems. Further, Seidewitz stated that a model's meaning has two aspects: one is the model's relationship to the things being modelled, and the other is about the properties and functions of the systems being modelled. In our previous work [1], we have demonstrated that these two aspects of semantics of models can be specified and examined separately. The former is called the *descriptive semantics*, which describes the structure of a model's instances, thus can be used to check the instance-of relationship by examining the structure of the system against the model. The latter is called the *functional semantics*, which focuses on the functionality and behaviour of the system being modelled. For example, consider the UML class diagram CD_1 depicted in Fig. 1. Informally, from the descriptive point of view, the semantics of the model is a set of statements such as

- *Person is a class*;
- *Woman is a class*; and
- *Woman is subclass of Person*.

These statements can be formally represented in predicate logic formulas as *Class*(*Person*), *Class*(*Woman*), and *Inherits*(*Woman*, *Person*), respectively. To judge whether a give system $\boldsymbol{S}$ (such as a program written in Java) is an instance of the model, we evaluate whether the following is true:



(a) Class Diagram CD_1 (b) Java Program Skeleton

Fig. 1 An M_1 model

$$\boldsymbol{S} \models Class(Person) \wedge Class(Woman) \wedge Inherits(Woman, Person).$$

It can be evaluated without referring to the behaviour of class and the properties of subclass/inheritance relation. For example, consider the Java program skeleton given in Fig. 1(b). We can judge that it is an instance of the model by recognizing that Person is a class and Woman is also a class and there is an inheritance relation from Woman to Person.

From functional semantics point of view, the semantics of the model CD_1 in Fig. 1

contains a set of statements like the following:

- *Person is a set of objects*;
- *Woman is a set of objects*;
- *Any object of Woman is also an object of Person*, etc.

Using predicate logic formulas, we write *Person*(x) to represent '*object x is a Person*', and *Woman*(x) for "*object x is a Woman*". Then, the third statement above can be represented formally as $\forall x.\ (Woman(x) \rightarrow Person(x))$. This statement imposes a constraint on the dynamic behaviour of a system, e.g. a Java program, as an instance of the model. From the functional semantics point of view, the model depicted in Fig. 1 also contains many other statements. For example, it also states that any attribute of *Person* is also an attribute of *Woman*. Here we only give some examples of such statements for the purpose of illustration.

In [1], we have developed a formal descriptive semantics of UML by defining mappings from UML models into predicate logic. As shown in Fig. 2, the mappings consist of the following sets of rules:

- *Sig*: signature mapping, which maps a metamodel N to a set of unary and binary predicate symbols and constant symbols. These symbols form a signature of predicate logic language.
- *Axm*: axiom mapping, which maps a metamodel N into a set of formulas over the signature *Sig*(N). *Axm*(N) represents the functional semantics of metamodel N, which is a set of statements must be satisfied by the models as instances of N.
- *Sem*: semantic mapping, which maps a model D into a set of formulas over the signature *Sig*(N), where N is a metamodel of D. *Sem*(D) describes the content of D in terms of types of the elements in D and relationships between the elements.
- *Hyp*: hypothesis mapping, which maps a model D into a set of formulas over *Sig*(N) , where N is the metamodel of D. *Hyp*(D) represents the hypothesis on how D is interpreted in a specific context.

Given a model D as an instance of metamodel N, the descriptive semantics of D is the set $Axm(N) \cup Sem(D) \cup Hyp(D)$ of formulas over signature *Sig*(N).

The above mappings have been implemented in a prototype tool called LAMBDES, which is integrated with a theorem prover SPASS [6] to enable automated reasoning about models. The mappings have been successfully applied to the class diagram, sequence diagram and state machine diagram of UML. Descriptive semantics of UML models has been used to check the consistency of models [4], to recognise design patterns in models and

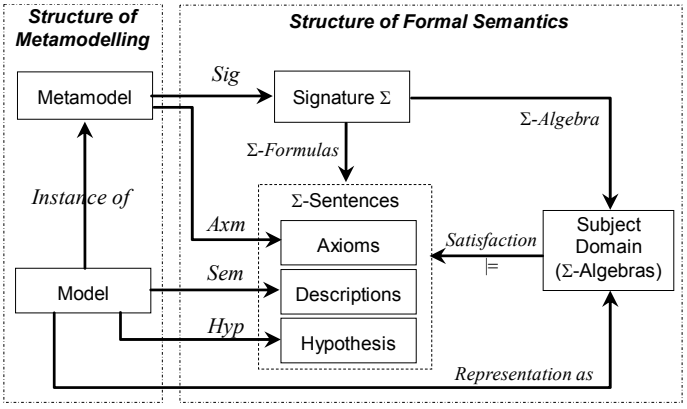


Fig. 2 Framework of the formal descriptive semantics [1]

to analyse relationships between design patterns [1]. The axiom mapping on metamodels has been used to check if a metamodel is well-defined in the sense that it is logically consistent [7] and if constraints imposed on M_1 models (e.g. well-formedness rules in OCL defined in the UML documentation or additional consistency rules) are valid in the sense that they are logically consistent with the metamodel.

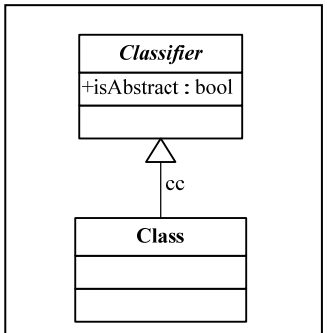


Fig. 3 A M_2 model CD_2

In this paper, we demonstrate that the above view to the semantics of models can equally be applied to models at other layers in the multi-layer metamodel hierarchy. In particular, at M_2 layer, OO concepts such as class and property are also used to classify elements in a metamodel, but called metaclass and meta-property respectively to avoid confusion. Take the class diagram CD_2 in Fig. 3 as an example. From the descriptive semantics point of view, the statements of the metamodel include:

- *Classifier is a metaclass*;
- *Class is a metaclass*; and
- *Class inherits Classifier*.

The above statements of the metamodel can be formalised as the following set of formulas.

{*MetaClass*(*Classifier*), *MetaClass*(*Class*), *Inherits*(*Class*, *Classifier*)}

From the functional semantics perspective, the inheritance arrow from metaclass *Class* to *Classifier* states that any instance of *Class* is also an instance of *Classifier*. This can be formalised as follows.

$$\forall x.(Class(x) \rightarrow Classifier(x))$$

The two aspects of semantics reveal that a metamodel in the multi-layer hierarchy plays two roles:

- *As an abstract syntax*, it defines the structure of its instances. In a M_i model (i>1), classes define element types in the instances of the model, and properties of the classes define inter-element relationships in the instances. This aspect is captured by the *descriptive semantics*, which specifies the element types and the relationships defined in a model with a set of first order formulas. The descriptive semantics of a model can be used to check if a system is a model's instance by examining whether the types of the elements in the system and their relationships are valid with respect to the model.
- *As an ontological semantics*, it defines a conceptual model of its instances. A M_i model (i>1), which defines a modelling language or a meta-modelling language, specifies the basic concepts underlying the language and the relationships between the concepts. Hence, it can be regarded as defining the ontology underlying the language. The *functional semantics* further characterises the basic concepts and their relationships by a set of axioms about their properties.

It is worth noting that we recognise the existence of semantic information contained in class diagrams when used as metamodels rather than merely abstract syntax. We argue that a class diagram depicts an ontology or a conceptual model of the subject domain. Viewing a metamodel as an ontology implies that the metamodel contains important semantic information, though an ontology is far from complete to define the semantics of a modelling language. For example, in the UML metamodel, a metaclass named *Class* refers to the notion of *class* in object-oriented software development paradigm. If the name is changed to something else, e.g. 'box', it no longer refers to the notion of class in object-orientation,

though the abstract syntax of UML is unchanged.

Concerning the whole multi-layer metamodel hierarchy, we propose a semantic framework for unifying the semantics of models at different layers. In this framework, the key question 'what is the instance-of relation?' is answered in the following way.

First, we formally define the semantics of a M_1 model M as a set of statements. In the sequel, we will write $[\![M]\!]$ to denote the set of formulas that the model M states. Thus, the 'instance-of' question is equivalent to 'whether a system satisfies the statements of the model'. In other words, a systems S is an instance of model M, if S satisfies the statements of M. In particular, as discussed above, the statements that a model makes are represented as a set of formulas in a predicate logic language, whose signature Σ is determined by its metamodel N. Moreover, we will divide the set $[\![M]\!]$ into two subsets: $[\![M]\!]_{Des}$ for the descriptive semantics, and $[\![M]\!]_{Fun}$ for the functional semantics. We will present rules to derive the sets of formulas $[\![M]\!]_{Des}$ and $[\![M]\!]_{Fun}$ from a class diagram M.

Second, we use mathematical structures (called *algebras* for short) of certain signature as abstract representations of systems in a subject domain. Therefore, the subject domain of a modelling language can be defined as a set of mathematical structures in the signature of the language. By doing so, the model theory of mathematical logics can be applied to formally define the satisfaction relationship |= between a system S and a model M. Therefore, system S is an instance of model M can be formally defined as $S \models [\![M]\!]$. What's important is that any M_i model (for all $i>0$) can be regarded as an algebra, too. Thus, the subject domains of models at all layers is unified at a high level of abstraction.

Consequently, the statement 'a M_i model M is an instance of a M_{i+1} model N' can also be formally translated into $M \models [\![N]\!]$. When separating descriptive from functional semantics, this is equivalent to $M \models [\![N]\!]_{Des} \cup [\![N]\!]_{Fun}$; or equivalently, $M \models [\![N]\!]_{Des}$ and $M \models [\![N]\!]_{Fun}$. The former holds if M is an Σ-algebra, where $\Sigma = Sig(N)$. This can be checked by parsing M according to N.

Moreover, we represent the descriptive semantics $[\![M]\!]_{Des}$ of M in the form of a set of logic formulas that characterises the mathematical structures of its instances. It is observed that the model M itself is also in that structure. Thus, the correctness of the definition of descriptive semantics of class diagrams can be expressed as $M \models [\![M]\!]_{Des}$. We prove the correctness of the rules to derive $[\![M]\!]_{Des}$ from M in this paper.

Furthermore, we define the functional semantics of UML class diagrams by a set of axioms that characterises the concepts of object-orientation underlying UML class diagrams. These axioms are represented in the form of higher order predicate logic formulas. It is observed that this set of axioms is independent of the usage of the class diagram, thus they are applicable to models and metamodels at all layers of the metamodel hierarchy.

Finally, the descriptive semantics and functional semantics are integrated through a set of rules that derive a set of first order logic formulas from models that all its instances must satisfy. We prove that the rules are correct in the sense they can be deduced from the functional and descriptive semantics.

The key feature of our approach is that the semantics of models/metamodels at different layers is unified into one theory, where the mappings from M to $[\![M]\!]_{Des}$ and $[\![M]\!]_{Fun}$ is invariant to the layer in which the model is interpreted, and the definitions of the concepts of metamodel hierarchy are identical for all layers.

3. Basic Concepts of Metamodel hierarchy

In this section, we define the basic concepts of metamodel hierarchy. We start with the concept of signatures of predicate logic languages and mathematical structures, and present

a set of rules to derive signatures from models at all layers in the UML metamodel hierarchy. Then, we define the notion of subject domain of models and modelling languages, etc. Finally, we define the concept of instance-of relation.

3.1. Signature

Let's first review the notion of signatures of predicate logic languages in which formulas are written.

Definition 1. (Signature)

The signature Σ of a predicate logic language *PrL* consists of three disjoint finite sets of symbols: a set Σ^0 of constant symbols, a set Σ^1 of unary predicate symbols, and a set Σ^2 of binary predicate symbols. □

In general, a signature of predicate logic language may also contain *N*-ary (*N*=3, 4, …) predicate symbols and function symbols. But, we will not use them in this paper.

Given an UML class diagram *D*, we define the signature derived from *D*, written *Sig*(*D*), through the following set of three signature rules SR_0, SR_1, and SR_2. They derive constant, unary predicate and binary predicate symbols from a class diagram, respectively. In the following discussion, we assume that a UML class diagram *D* is a M_{i+1} model, i.e. the metamodel of some M_i models, where $i \geq 0$.

An enumeration class in *D* defines a data type whose values are the enumeration literals. We use a constant symbol to represent an enumeration value. Let *D.EnumValue* denote the set of enumeration values in *D*. Hence, we have the following signature rule.

Rule SR0. (Constants)

For each enumeration value *V* given in an enumeration class *E* in *D*, we include a constant symbol *V* in Σ^0. Formally, $\Sigma^0 = SR_0(D) = \{V \mid V \in D.EnumValue\}$. □

For example, Fig. 4shows examples of models at different layers in the metamodel

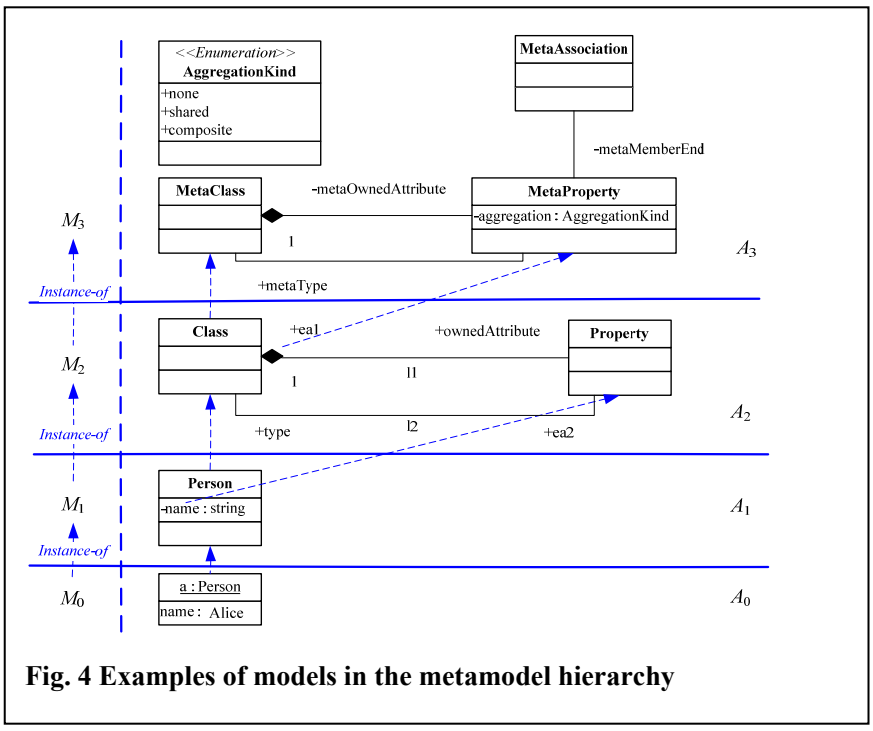


Fig. 4 Examples of models in the metamodel hierarchy

hierarchy, where A_0 partly depicts a snapshot of a run-time program, A_1 is a user-defined UML model, A_2 is a subset of UML metamodel, and A_3 is a subset of the MOF model. The enumeration class *AggregationKind* in A_3 defines a data type for the attribute *aggregation* of class *MetaProperty*. By applying SR_0 on A_3, we obtain constant symbols *none*, *shared* and *composite* from the enumeration values of *AggregationKind*.

A class in model D is a classification of elements in an instance of D. Let $D.Class$ denote the set of classes in D. Thus, we have the following signature mapping.

Rule SR1. (Unary predicate symbols)

For each class named C in D, we include a unary predicate symbol C in $\Sigma^1 \subseteq Sig(D)$. Formally, $\Sigma^1 = SR_1(D) = \{C \mid C \in D.Class\}$. □

Informally, for an element x in a M_i model M, $C(x)$ means that element x has type C. For example, given class diagrams in Fig. 4 , by applying rule SR_1 to A_1, we obtain a unary predicate symbol *Person*(x). The formula *Person*(*Alice*) means that the element *Alice* in A_0 is of type *Person*. By applying SR_1 on A_2, we derive two unary predicates symbols *Class*(x) and *Property*(x). Formula *Class*(*Person*) means that the element *Person* in A_1 is a *class*; and *Property*(*name*) means that *name* is a *property*. By applying SR_1 on A_3, we derive unary predicate *MetaClass*(x), *MetaAssociation*(x), *MetaProperty*(x) and *AggregationKind*(x). Then formulas *MetaClass*(*Class*), *MetaClass*(*Property*), *MetaAssociation*(*l1*), *MetaAssociation*(*a2*), *MetaProperty*(*type*) and *MetaProperty*(*ownedAttribute*) assert the types of elements in model A_2.

In class diagram D, an association between classes X and Y with label A on the association end at Y's side defines a relationship A that instances of X and instances of Y may hold in an instance model of D. An attribute A of X with Y as the data type also defines such a relationship. Let $D.Property$ and $D.AssociationEnd$ denote the set of properties and association ends in D, respectively. We use a binary predicate to represent a relationship, hence the following signature rule.

Rule SR2 (Binary predicates).

For each attribute R of class X with class Y as the data type, and each association from class X to class Y with R as the association end in D, we include a binary predicate symbol R *in* Σ^2. Formally, $\Sigma^2 = SR_2(D) = \{R \mid R \in D.Property \vee R \in D.AssociationEnd\}$. □

Informally, for a pair of elements (x, y), $R(x, y)$ means that there is an R relationship between x and y. For example, by applying SR_2 on A_1 in Fig. 4, we obtain a binary predicate symbol *name*(x, y). The formula *name*(p, *Alice*) means that the value of the attribute *name* of p in A_0 is *Alice*. By applying SR_2 on A_2, we obtain binary predicate symbols *ownedAttribute*(x, y) and *type*(x, y). The attribute definition '*name*: *string*' in A_1 can be described as *ownedAttribute*(*Person*, *name*) and *type*(*name*, *string*). By applying SR_2 on A_3, we obtain binary predicate symbols *aggregation*(x, y), *metaType*(x, y) and *metaMemberEnd*(x, y). The statements about the attribute *aggregation* of the association ends of association *l1* in A_1 can be stated as *aggregation*(*el1*, *composite*) and *aggregation*(*ownedAttribute*, *none*).

Definition 2 (Signature induced from metamodel)

Let D be a UML class diagram. We define $\Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 = Sig(D) = SR_0(D) \cup SR_1(D) \cup SR_2(D)$ to be the signature induced from D, where $\Sigma^i = SR_i(D)$, $i = 0, 1, 2$. □

3.2. Subject Domain

We use mathematical structures to represent systems in subject domains at all layers of the UML metamodel hierarchy. Given a signature Σ, we call such mathematical structures

Σ-algebras.

Definition 3. (Σ-Algebra)

Let $\Sigma=\Sigma^0\cup\Sigma^1\cup\Sigma^2$ be any given signature. An Σ-algebra $\mathscr{A}=(A, Pr, Rel)$ is a mathematical structure where A is an non-empty set, called the carrier set, Pr is a set of unary predicates on A, and Rel is a set of binary predicates on A, such that,

- for each constant symbol $c\in\Sigma^0$, there is a corresponding element $c_{\mathscr{A}}\in A$;
- for each unary predicate symbol $P\in\Sigma^1$, there is a corresponding unary predicate $P_{\mathscr{A}}\in Pr$;
- for each binary predicate symbol $R\in\Sigma^2$, there is a corresponding binary predicate $R_{\mathscr{A}}\in Rel$. □

Given a signature Σ, to represent a system S as a Σ-algebra $\mathscr{A}_S$, we first consider each unary predicate symbol P in Σ as representing a type of elements. We identify the elements in the system S that are regarded as of type P. The set of such elements identified for all unary predicates in Σ forms the carrier set A of the algebra $\mathscr{A}_S$. The unary predicate $P_{\mathscr{A}}$ in $\mathscr{A}_S$ corresponding to symbol P is defined such that $P_{\mathscr{A}}(a)$ is true for an element $a\in A$ if and only if the element a is of type P. Each binary predicate symbol R in Σ is regarded as representing a relation on the elements in the system. The corresponding relation $R_{\mathscr{A}}$ in $\mathscr{A}_S$ is defined such that $R_{\mathscr{A}}(a, b)$ is true for elements $a,b\in A$ if and only if the relation holds between these two elements in the system S. In the case that the carrier set A is the empty set ∅, the algebra $\mathscr{A}_S$ is trivial. This indicates that the system cannot be meaningfully represented as a Σ-algebra.

Example 1. (Program as algebra)

Let signature Σ be $\Sigma^0\cup\Sigma^1\cup\Sigma^2$, where $\Sigma^0=\varnothing$, $\Sigma^1=\{Class, Attribute\}$, and $\Sigma^2=\{Inherits, HasAttribute\}$. Consider the Java program skeleton given in Fig. 1(b). We can represent it as the following algebra, which is referred to as Alg_1 in the sequel.

$A=\{Person, Woman, Name\}$

$Class(Person)=true$, $Class(Woman)=true$, $Class(Name)=false$;

$Attribute(Person)=false$, $Attribute(Woman)=false$, $Attribute(Name)=true$;

$Inherits(Woman, Person)=true$;	$HasAttribute(Woman, Person)=false$;
$Inherits(Woman, Name)=false$;	$HasAttribute(Woman, Name)=true$;
$Inherits(Woman, Woman)=false$;	$HasAttribute(Woman, Woman)=false$;
$Inherits(Person, Woman)=false$;	$HasAttribute(Person, Woman)=false$;
$Inherits(Person, Name)=false$;	$HasAttribute(Person, Name)=true$;
$Inherits(Person, Person)=false$;	$HasAttribute(Person, Person)=false$;
$Inherits(Name, Person)=false$;	$HasAttribute(Name, Person)=false$;
$Inherits(Name, Woman)=false$;	$HasAttribute(Name, Woman)=false$;
$Inherits(Name, Name)=false$;	$HasAttribute(Name, Name)=false$.

The mathematical structure Alg_1 satisfies the statements $Class(Person)$, $Class(Woman)$ and $Inherits(Woman, Person)$ in the descriptive semantics of the model CD_1. □

Extracting algebraic structural information from program source code has been implemented by various reverse engineering tools such as those used to recover design patterns in software [8].

Similarly, information contained in graphic models can also be represented as algebras following the same procedure described above for extracting algebraic structural information from software systems.

Example 2. (Model as algebra)

Let signature Σ' be $\Sigma'^0\cup\Sigma'^1\cup\Sigma'^2$, where $\Sigma'^0=\varnothing$, $\Sigma'^1=\{MetaClass,$

MetaRelation}, $\Sigma'^{2}=\varnothing$. Here, we interpret the unary predicate symbol *MetaClass* as the type of the element types in the model, and the unary predicate symbol *MetaRelation* as the type of the relations between the elements in the model. Therefore, the 'elements' x in CD_1 such that *MetaClass*(x) is true are *Class*, and *Attribute*. The 'elements' x in CD_1 such that *MetaRelation*(x) is true are *Inherits* and *HasAttribute*. The carrier set is, therefore, {*Class*, *Attribute*, *Inherits*, *HasAttribute*}. The class diagram CD_1 given in Fig. 1(a) can thus be represented as the following algebra, which is referred to as Alg_2.

A = {*Class*, *Attribute*, *Inherits*, *HasAttribute*}.

MetaClass(*Class*)=*true*, *MetaRelation*(*Class*)=*false*,
MetaClass(*Attribute*)=*true*, *MetaRelation*(*Attribute*)=*false*,
MetaClass(*Inherits*)=*false*, *MetaRelation*(*Inherits*)=*true*,
MetaClass(*HasAttribute*)=*false*; *MetaRelation*(*HasAttribute*)=*true*;

The mathematical structure Alg_2 can be used to evaluate the truth of descriptive statements at metamodel level, such as *MetaClass*(*Class*) ∧*MetaClass*(*Attribute*).

□

It is worth noting that the representation of a model or system as an algebra depends on the signature and the semantics interpretation of the symbols in the signature. One system or model can be represented differently when the signature is different or the interpretation of the symbols is different. For example, consider the class diagram CD_1 depicted in Fig. 1. Given the signature is Σ in Example 1, we can interpret the unary predicate symbol *Class* as the class nodes in a class diagram, the unary predicate symbol *Attribute* as the items in the attribute compartments of class nodes, etc. Consequently, the model CD_1 can be represented exactly the same as the algebra Alg_1. Being able to represent both the class diagram CD_1 and the Java program skeleton given in Fig. 1 as the same algebra Alg_1 reflects the fact that the Java program is an instance of the model CD_1 in the context of the signature Σ.

Definition 4. (Subject domain)

Let class diagram D be a M_i model and Σ=*Sig*(D) the signature induced from D. The collection of all Σ-algebra is called the *immediate subject domain of model D*, denoted by $Dom^{<i>}(D)$. For i≥1, the *ultimate subject domain* of a M_i model D, denoted by $Dom^*(D)$ is inductively defined as follows.

For i=1, $Dom^*(D)=Dom^{<1>}(D)$.

For i>1, $Dom^*(D)=\bigcup\left\{Dom^*(D_x)\middle|D_x \in Dom^{<i>}(D)\right\}$ □

For example, let U denote the metamodel of the UML language and $\boldsymbol{N}$ the set of all UML models. U is a M_2 model. As illustrated by Example 1, any UML model can be represented as a mathematical structure in the signature induced from U. Therefore, the immediate subject domain of the model U is the set of all UML models, i.e. $Dom^{<2>}(U) = \boldsymbol{N}$. For a M_1 model $U_x \in \boldsymbol{N}$, its immediate subject domain $Dom^{<1>}(U_x)$ is the collection of all mathematical structures in the signature induced from U_x, including the OO programs whose static structures are captured by U_x. Therefore, the ultimate subject domain of UML contains all OO programs written in Java or any object-oriented programming languages, i.e.

$$SubDom(U)=\bigcup\left\{SubDom(U_x)\middle|U_x \in SubDom^{<2>}(U)\right\} = \bigcup\left\{SubDom(U_x)\middle|U_x \in N\right\}$$

In general, for a M_2 model D which defines a modelling language L, its immediate subject domain contains all models in L, and its ultimate subject domain contains all mathematical structures in the signatures induced from models in L. For the only M_3 model MOF, since any M_2 model is an instance of MOF and can be represented as a mathematical structure induced from MOF, its immediate subject domain is the set of all M_2 models. The

ultimate subject domain of MOF contains the subject domains of all M_2 model.

3.3. Instance-of Relation

Informally, a system S is an instance of a model M, if (a) S is in the subject domain of the model, i.e. the system S can be represented as a mathematical structure in the signature induced from M; and, (b) S satisfies the statements of M. The following defines the syntax of the formulas representing statements of models.

Definition 5 (Formulas)

Given a signature $\Sigma=\Sigma^0\cup\Sigma^1\cup\Sigma^2$, and a collection $V=V^0\cup V^1\cup V^2$ of disjoint and countable sets of variables, the predicate logic formulas are inductively defined as follows.

- For all unary predicate symbols $P\in\Sigma^1$, constant symbols $c\in\Sigma^0$, and variables $x\in V^0$, $P(c)$ and $P(x)$ are formulas;
- For all binary predicate symbols $R\in\Sigma^2$, constant symbols $c_1, c_2 \in \Sigma^0$, and variables $x_1, x_2 \in V^0$, $R(c_1, c_2)$, $R(x_1, c_2)$, $R(c_1, x_2)$ and $R(x_1, x_2)$ are formulas;
- For all variables $X\in V^1$, constant symbols $c\in\Sigma^0$, and variables $x\in V^0$, $X(c)$ and $X(x)$ are formulas;
- For all variables $X\in V^2$, constant symbols $c_1, c_2 \in\Sigma^0$, and variables $x_1, x_2 \in V^0$, $X(c_1, c_2)$, $X(x_1, c_2)$, $X(c_1, x_2)$ and $X(x_1, x_2)$ are formulas;
- $F_1 \wedge F_2$, $F_1 \vee F_2$, $F_1 \Rightarrow F_2$, $F_1 \Leftrightarrow F_2$, $\neg F_1$ are formulas, If F_1 and F_2 are formulas;
- $\forall x.F$ and $\exists x.F$ are formulas, if $x \in V$ and F is a formula.

In the sequel, we write $Formula(\Sigma,V)$ to denote the set of formulas in signature Σ with variables in V. □

Let $\mathscr{A}$ be a Σ-algebra, an assignment α of variables V to Σ-algebra $\mathscr{A}$ is a mapping from V to $\mathscr{A}$ such that

(a) for each $x\in V^0$, $\alpha(x) \in A$;
(b) for each $x\in V^1$, $\alpha(x) \in Pr$; and
(c) for each $x\in V^2$, $\alpha(x) \in Rel$.

Given a Σ-algebra $\mathscr{A}$, a formula ϕ, and an assignment α of variables in ϕ, we define $Eva_{\mathscr{A},\alpha}(\phi)$ as follows.

- $Eva_{\mathscr{A},\alpha}(P(c))=True$, if and only if $P_{\mathscr{A}}(c_{\mathscr{A}})$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(P(x))=True$, if and only if $P_{\mathscr{A}}(\alpha(x))$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(R(c_1, c_2))=True$, if and only if $R_{\mathscr{A}}(c_{1\mathscr{A}},c_{2\mathscr{A}})$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(R(x_1, c_2))=True$, if and only if $R_{\mathscr{A}}(\alpha(x_1), c_{2\mathscr{A}})$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(R(c_1, x_2))=True$, if and only if $R_{\mathscr{A}}(c_{2\mathscr{A}}, \alpha(x_2))$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(R(x_1, x_2))=True$, if and only if $R_{\mathscr{A}}(\alpha(x_1), \alpha(x_2))$ is true in $\mathscr{A}$;
- $Eva_{\mathscr{A},\alpha}(X(c))=True$, if and only if $P_{\alpha}(c_{\mathscr{A}})$ is true in $\mathscr{A}$, where $P_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(X(x))=True$, if and only if $P_{\alpha}(\alpha(x))$ is true in $\mathscr{A}$, where $P_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(X(c_1, c_2))=True$, if and only if $R_{\alpha}(c_{1\mathscr{A}},c_{2\mathscr{A}})$ is true in $\mathscr{A}$, where $R_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(X(x_1, c_2))=True$, if and only if $R_{\alpha}(\alpha(x_1), c_{2\mathscr{A}})$ is true in $\mathscr{A}$, where $R_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(X(c_1, x_2))=True$, if and only if $R_{\alpha}(c_{2\mathscr{A}}, \alpha(x_2))$ is true in $\mathscr{A}$, where $R_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(X(x_1, x_2))=True$, if and only if $R_{\alpha}(\alpha(x_1), \alpha(x_2))$ is true in $\mathscr{A}$, where $R_{\alpha}=\alpha(X)$;
- $Eva_{\mathscr{A},\alpha}(F_1 \wedge F_2)=True$, if and only if $Eva_{\mathscr{A},\alpha}(F_1)=True$ and $Eva_{\mathscr{A},\alpha}(F_2)=True$;
- $Eva_{\mathscr{A},\alpha}(F_1 \vee F_2)=True$, if and only if $Eva_{\mathscr{A},\alpha}(F_1)=True$ or $Eva_{\mathscr{A},\alpha}(F_2)=True$;
- $Eva_{\mathscr{A},\alpha}(F_1 \Rightarrow F_2)=True$, if and only if $Eva_{\mathscr{A},\alpha}(F_1)=False$ or $Eva_{\mathscr{A},\alpha}(F_2)=True$;
- $Eva_{\mathscr{A},\alpha}(F_1 \Leftrightarrow F_2)=True$, if and only if $Eva_{\mathscr{A},\alpha}(F_1)=Eva_{\mathscr{A},\alpha}(F_2)$;
- $Eva_{\mathscr{A},\alpha}(\neg F)=True$, if and only if $Eva_{\mathscr{A},\alpha}(F)=False$;
- $Eva_{\mathscr{A},\alpha}(\forall x.F)=True$, if and only if $Eva_{\mathscr{A},\alpha'}(F)=True$ for all assignment α' such that $\alpha'(z)=\alpha(z)$ for all $z\neq x$;

– $Eva_{\mathcal{A},\alpha}(\exists x.F)=True$, if and only if $Eva_{\mathcal{A},\alpha'}(F)=True$ for at least one assignment α' such that $\alpha'(z)=\alpha(z)$ for all $z\neq x$.

Definition 6 (Satisfaction relation)

Let Σ be any given signature, $\mathcal{A}$ be a Σ-algebra and F be a formula in signature Σ. We say that $\mathcal{A}$ satisfies F, written $\mathcal{A} \models F$, if there is an assignment α in $\mathcal{A}$ such that $Eva_{\mathcal{A},\alpha}(F)=True$.

Let Φ be a set of formulas in signature Σ. We say that $\mathcal{A}$ satisfies Φ, write $\mathcal{A} \models \Phi$, if for all $F \in \Phi$, we have that $\mathcal{A} \models F$. □

For example, by the above definition of |=, it is easy to see that the following statement is true in the algebra given in Example 2.

$$\forall x.HasAttribute(Person, x) \rightarrow HasAttribute(Woman, x).$$

Note that the above definition applies to models at all layers in the metamodel hierarchy.

By representing systems and models at all layers of metamodel hierarchy as mathematical structures and defining the semantics of models/metamodels as a set of statements, the instance-of relation between a structure and a model can be defined by employing the satisfaction relation.

Definition 7 (Instance-of relation)

Let M be a system/model at M_i level, and N be a M_{i+1} model, for $i \geq 0$. Let $Rep_\Sigma(M)$ denote the representation of M as a mathematical structure in the signature Σ and $Semantics_\Sigma(N)$ be the set of statements in the signature Σ that defines the semantics of N. We say that M is an ***instance of*** N, if

(a) $Rep_\Sigma(M)$ is a non-trivial Σ-algebra, where $\Sigma=Sig(N)$; and

(b) $Rep_\Sigma(M) \models Semantics_\Sigma(N)$.

For the sake of convenience, in the sequel we will also write $M \models_\Sigma \Phi$ to denote that the algebraic representation of model M in signature Σ satisfies the Σ statements in Φ. When there is no risk of confusion, we also omit the subscript Σ. □

In the following two sections, we will define the functional and descriptive semantics for UML models and metamodels at all layers.

4. Descriptive Semantics

In this section, we present a set of rules to derive a set of formulas from a class diagram to represent the descriptive semantics of the model. We will demonstrate that the rules are applicable to class diagrams used at all layers of the UML metamodel hierarchy. We will then identify the characteristics of using class diagrams in metamodelling and specify the usage context as a set of rules that derive additional descriptive statements from models.

4.1. Translation mapping

Given a M_{i+1} model N, and a M_i model D, the following set of translation mapping rules translate D into a set of formulas in the signature Σ determined by N. At the same time, the rules check if D is a Σ-algebra.

Rule TR1. (Classification of elements)

For each element a of type C in model D, a formula $C(a)$ is generated, if C is a concrete class in N. If D contains an element whose type is not a concrete class in N, D is not an instance of N. □

We write $R_N(M)$ to denote the set of statement generated from M_i model M based on M_{i+1}

model N by applying rule R. The subscript N may be omitted when there is no risk of confusion.

For example, by applying rule $TR1$ to the M_2 model A_2 in Fig. 4 based on the M_3 model A_3, the following formulas are derived, stating that *Class* and *Property* are instances of *MetaClass*, *a* is an instance of *MetaAssociation*, *ownedAttribute* and *owner* are instances of *MetaProperty*.

$$TR1_{A3}(A_2) = \{MetaClass(Class), MetaClass(Property),$$
$$MetaAssociation(l1), MetaAssociation(l2),$$
$$MetaProperty(ownedAttribute), MetaProperty(type)\ \}$$

By applying rule $TR1$ to the M_1 model A_1 in Fig. 4 and the UML metamodel, the following formulas can be derived, stating that *Person* is an instance of *Class* and *Name* is an instance of *Property*.

$$TR1_{A2}(A_1)=\{Class(Person), Property(Name)\}$$

Elements in a model bear certain relationship, which are mostly expressed through their relative position. In a class diagram, for instance, an attribute definition inside a class node indicates that the class owns this attribute. Such implicitly specified relationships in a model should be explicitly expressed in the descriptive semantics of the model. Hence we have the following mapping rule.

Rule TR2. (Relationships between elements)

For a pair (e_1, e_2) of elements in a model M which has relationship R, a formula in the form of $R(e_1, e_2)$ is generated, if R is a meta-relation (i.e. either a meta-attribute or a meta-association in the metamodel N. If R is not a meta-relation in the metamodel N, the model is not an instance of the metamodel N. □

For example, by applying rule $TR2$ to A_2 in Fig. 4 and the M_2 model A_3 as the metamodel, the following formulas can be derived.

$$TR2_{A3}(A_2)=\{metaMemberEnd(l1, ownedAttribute), metaMemberEnd(a2, type),$$
$$metaType(ownedAttribute, Property), metaType(type, Class)\}$$

By applying rule $TR2$ to A_1 in Fig. 4using the UML metamodel as the metamodel, the following formulas can be derived.

$$TR2_{A2}(A_1)=\{ownedAttribute(Person, Name), type(Name, String)\}$$

Note that the translation rules play two roles. First, it derives a set of formulas as a part of the descriptive semantics of the model. Second, it checks if a model is in the structure required by its metamodel. If the formulas are successfully generated, then it is a Σ-algebra. Thus, it satisfies the descriptive semantics of its metamodel.

It is also interesting to observe that a M_3 model can be used as its own metamodel when applying the translation rules. For example by applying rule $TR2$ to A_3 in Fig. 4 using A_3 itself as the metamodel, the following formulas are derived.

$$TR2_{A3}(A_3) = \{\ meta\text{-}metaOwnedAttribute(MetaProperty, aggregation),$$
$$meta\text{-}metaType(aggregation, AggregationKind),$$
$$meta\text{-}metaMemberEnd(k1, metaOwnedAttribute),$$
$$meta\text{-}metaType(metaOwnedAttribute, MetaProperty),$$
$$meta\text{-}metaMemberEnd(k2, metaType),$$
$$meta\text{-}metaType(metaType, MetaClass),$$
$$meta\text{-}metaMemberEnd(k3, metaMemberEnd),$$

$$meta\text{-}metaType(metaMemberEnd, MetaProperty)\}.$$

where, to avoid naming confliction and confusion, a prefix '*meta-*' is added to each symbol of A_3 when it is used as the metamodel. This reveals the reflection of the M_3 model, which "extends a model with the ability to be self describing" [3].

In the sequel, we write $TR_N(M)$ to denote the set of statements generated from M_i model M based on M_{i+1} model N by applying rule *TR1* and *TR2*. We also often omit the subscript N when there is no risk of confusion. That is,

$$TR_N(M) = TR1_N(M) \cup TR2_N(M) .$$

It is easy to see that the translation mapping is complete in the sense that every element and relationship in a M_i model is represented in the generated formulas. The following theorem proves that the above rules are correct as the descriptive semantics of class diagrams.

Theorem 1. (Correctness of the descriptive semantics mappings)

Let N be a M_{i+1} model and M be a M_i model. If model M is a valid instance of N, then the following two statements are true.

(a) The formulas generated are syntactically valid. Formally,

$$TR_N(M) \subseteq Formula(Sig(N), \varnothing).$$

(b) The model M's structure is reflected in the generated formulas. Formally,

$$M \models TR_N(M).$$

Proof.

(a) We prove statement (a) via contradiction.

Assume there is a formula ϕ such that $\phi \in TR_N(M)_s$, but $\phi \notin Formula(Sig(N),\varnothing)$. If ϕ is generated by applying rule *TR*1, then, according to the definition of *TR*1, there is an element a in model M of type C such that $\phi = C(\alpha)$. Since, $\phi \notin Formula(Sig(N),\varnothing)$, we have that C is not a class in metamodel N, according to *SR*1. Therefore, there is a model element in M that does not belong to a class in the metamodel. Thus, M is not a valid instance of N. This contradicts the condition of the theorem. Similarly, if ϕ is generated by applying rule *TR*2, we have that there are elements a_1 and a_2 in model M that are related by a relation R and $\phi = R(a_1, a_2)$. Because $\phi \notin Formula(Sig(N),\varnothing)$, according to the signature mapping rule *SR*2, R is not an attribute or association in the metamodel N. Therefore, elements a_1 and a_2 in model M cannot be related in a valid instance model of N. This contradicts the condition of the theorem. In conclusion, the assumption that $\phi \notin Formula(Sig(N),\varnothing)$ is not true. Thus, the statement (a) holds.

(b) Now, we prove statement (b).

Let $\phi \in TR_N(M)$. If ϕ is generated by applying rule *TR*1, according to the definition of *TR*1, there is an element a in model M of type C such that $\phi = C(\alpha)$. Because $C(\alpha) \in Formula(Sig(N))$ according to statement (a) proved above, C is a unary predicate symbol in the signature of the algebraic representation of model M. Therefore, in the algebraic representation of M, we have $C(\alpha) = true$. Thus, $M \models \phi$. If ϕ is generated by applying rule *TR*2, we have that there are elements a_1 and a_2 in model M that are related by a relation R and $\phi = R(a_1, a_2)$. Similarly, we have that in the algebraic representation of M, we have the $R(a_1, a_2)$=true. Therefore, we also have that $M \models \phi$. Thus, statement (b) is also true.

□

Note that, the set of formulas derived from a class diagram using the translation rules allow flexibility in the interpretation of the diagrams differently according to the usage of the model. In the following subsection, we identify the usage context of class diagrams in metamodelling and specify the context in the form of a set of rules.

4.2. Hypothesis mapping

The interpretation of a UML model depends on the context in which the model is used. For example, a UML model may play the role of a sketch design of a program, which means each element in the model is supposed to have a corresponding element of the same type in the program, but the program may be allowed to have elements that are not depicted in the diagram. A model may also be used as a detailed design, which requires it to depict all classes in the program as well as all attributes and operations of the classes. Such assumptions on the relationship between a model and the modelled structures are not explicitly specified in the model, but are necessary when interpreting the model, therefore need to be formalised in descriptive semantics. Our approach is to allow the users to specify a set of hypothesis about the uses of the model in the form of logic formulas. In this section, we discuss the context of using class diagrams in metamodelling and define a set of hypothesis mappings that characterises the context.

Let e_1, e_2, …, e_k be the set of elements in a M_i model M, and C be their direct type.

Rule HR1. (Distinguishability of elements)

Elements of type C are all different. Thus, we have the following set of formulas:

$$\{e_i \neq e_j \mid \text{for } i \neq j \in \{1, 2, \ldots k\}\}. \quad \square$$

For example, the class diagram B_2 in Fig. 5 is a metamodel, and we expect that *Association* is different from *Generalisation* and *Class*, etc. By applying rule *HR*1 to class nodes in B_2, we obtain the following set of formulas.

$$HR1(B_2) = \{Association \neq Class, Association \neq Generalisation, Association \neq Classifier \ldots\}$$

When HR_1 is applied to classes in B_1, we obtain the following set of formulas.

$$HR1(B_1) = \{Person \neq Woman, Person \neq Man, Man \neq Woman\}$$

These formulas are necessary when B_1 is used as a model of the real world, where woman, man and person are different concepts. However, if B_1 is used as a requirements specification of a software system, these formulas may be unnecessary, because a program containing one class implementing *Person* with an attribute *Sex*, whose value is *Male* or *Female*, is also be a correct implementation of B_1. In this case, the hypothesis rule *HR*1 is not applicable.

Rule HR2. (Completeness of elements)

The set of elements of a type C is complete. Formally,

$$\forall x.\ (C(x) \rightarrow (x = e_1) \vee (x = e_2) \vee \ldots \vee (x = e_k)). \quad \square$$

For example, by applying rule *HR*2 to classes in B_2 in Fig. 5, we obtain the following formula.

$$\forall x.\ (MetaClass(x) \rightarrow (x = Association) \vee (x = Class)$$
$$\vee\ (x = Generalisation) \vee (x = Classifier))$$

This means in the modelling language specified by B_2, the metaclasses can only be *Association*, *Generalisation*, *Class* and *Classifier*. Therefore, elements in the instances of B_2 can only be of type Association, Generalisation, Class or Classifier.

When *HR*2 rule is applied to B_1, we obtain the following formula.

$$HR2(B_1) = \{\forall x.((Class(x) \rightarrow (x = Person) \vee (x = Man) \vee (x = Woman)))\}$$

This formula is not required if a program containing additional classes is regarded as a correct implementation of B_1. It is required when B_1 is used as a model derived from code in

reverse engineering.

Similarly, we have the following hypothesis on the completeness of relations in metamodels. Let $R(x_1, x_2)$ be a binary predicate, $R(e_{1,1}, e_{1,2})$, $R(e_{2,1}, e_{2,2})$, …, $R(e_{n,1}, e_{n,2})$ be the set of R relations explicitly depicted in the metamodel.

Rule HR3. (Completeness of relations)

Relation R is completely depicted in metamodels. Formally, we have the following formula:

$$\forall x_1, x_2.(R(x_1, x_2) \rightarrow ((x_1=e_{1,1})\wedge(x_2=e_{1,2}))\vee((x_1= e_{2,1})\wedge (x_2= e_{2,2}))\vee\ldots ((x_1= e_{n,1})\wedge (x_2= e_{n,2})))$$

□

This hypothesis states that all relations of a certain type are explicitly specified in metamodels, thus any additional relation in an instance model will be regarded as not satisfying the metamodel. For example, by applying *HR*3 to the relationship *specific* in B_2 in Fig. 5 , the following formula can be obtained.

$$\forall x, y.\ specific(x, y) \rightarrow ((x=cc) \wedge (y=Class)) \vee ((x=ac) \wedge (y= Association))$$

where *cc* is the identifier of the generalisation arrow from *Class* to *Classifier*, and *ac* the identifier of the generalisation arrow from *Association* to *Classifier*.

Again, this rule is not always applicable to models at layer M_1. If it is applied to the relationship *specific* in B_1 in Fig. 5, we obtain the following formula.

$$\forall x, y.\ specific(x, y) \rightarrow ((x=wp) \wedge (y=Person)) \vee ((x=mp) \wedge (y= Person))$$

where *wp* is the identifier of the generalisation arrow from *Woman* to *Person*, and *mp* the identifier of the generalisation arrow from *Man* to *Person*. This is not necessarily true, because, for example, there may be additional classes in the system and additional inheritance between them.

One of the most important *hypothesis* on the uses of class diagrams as metamodels is the *strict metamodelling principle*. It was proposed by Atkinson [9] to ensure that a metamodel is a well-defined abstract syntax of modelling language. The strict metamodelling principle states that:

"*In an n-level modelling architecture* M_0, M_1, …, M_n, *every element of an* M_i*-level model must be an* instance-of ***exactly one element*** *of an* M_{i+1}*-level model, for all* $0 \le i < n$-1."

Therefore, we have the following hypothesis rule, which asserts that an element is only in one concrete class.

Rule HR4. (Disjointness of classification)

Let C_1, C_2, …, C_n be the set of concrete classes in D. For each pair of different concrete classes C_i and C_j, $i \neq j$, we include the formula $\forall x.\ (C_i(x) \rightarrow \neg\ C_j(x))$ in $Axm(D)$. □

For example, by applying rule *HR*4 on B_2 in Fig. 5, the following statements are derived, which state that instances of *Class*, *Association* and *Generalisation* are disjoint with each other.

$$HR4(D_2) = \{\forall x.\ (Class(x) \rightarrow \neg\ Association(x)),$$
$$\forall x.\ (Association(x) \rightarrow \neg\ Generalisation(x)),$$
$$\forall x.\ (Class\ (x) \rightarrow \neg\ Generalisation(x))\ \}$$

Rule HR4 is sometimes applicable to models at M_1 level, but not always. For example, by applying *HR*4 on B_1, the following axioms on instances of B_1 are derived, which state that instances of *Man* and *Woman* are disjoint.

$$HR4(D_1) = \{\forall x.\ (Woman(x) \rightarrow \neg\ Man(x))\}$$

In this case, it is true. However, the rule is not always applicable, especially when '*multiple*

inheritances' is allowed.

Note that the above rules are also based on a metamodel N to determine the type an element belongs to. We have omitted this issue in the above discussion for the sake of readability. In the sequel, we write $HR_N(M)$ to denote the set of statements generated from a model M according to a metamodel N by applying the above hypothesis rules.

To conclude this section, we now formally define the descriptive semantics of metamodels at all layers as the set of statements generated by the translation rules and hypothesis rules.

Definition 8 (Descriptive semantics)

Given a M_i model M as an instance of metamodel N at M_{i+1} level, the descriptive semantics of M, written $\Box M \Box_{Des}$, is defined to be the set of formulas $TR_N(M) \cup HR_N(M)$. □

5. Functional Semantics

As discussed in Section 2, the functional semantics of UML defines the properties of the basic OO concepts underlying the language. In general, functional semantics include both static and dynamic semantics, where the former are time invariant and/or time independent features, while the latter are the temporal aspect of functionality and behaviour. Since models are static, i.e. the set of statements that a model states are invariant of time, the functional semantics of metamodels only involves static functional semantics. Thus, in this paper we only give the static functional semantics of metamodels.

We specify the OO concepts by a set of axioms in second order predicate logic. The predicates used in the axioms, except for the additionally defined ones, are from the signature induced from the UML metamodel. These axioms are applicable to all models and systems at all levels.

The static functional semantics for OO systems consists of the following axioms.

5.1. Basic Axioms

The first group of axioms are about the basic properties of classes and objects.

Axiom 1. (Classification of objects)

Every object must be an instance of a class. Formally,

$$\forall x .(Object(x) \rightarrow \exists C.(Class(C) \wedge C(x))).$$ □

Axiom 2. (Attribute declarations)

Every attribute declared in a class is a property of the class. Let *HasAttribute* be a binary predicate. Formally, we have that

$$\forall x .\forall C .(Class(C) \wedge Property(x) \wedge OwnedAttribute(C, x) \rightarrow HasAttribute(C, x))$$ □

Axiom 3. (Operations declarations)

Every operation declared in a class is an operation of the class. Formally,

$$\forall x .\forall C .(Class(C) \wedge Operation(x) \wedge OwnedOperation(C, x) \rightarrow HasOperation(C, x))$$ □

The following axiom is about the composition relation.

Axiom 4. (Composite relation)

Assume that there is a composite relation from class A to class B (i.e. B is a part of A). For each object x in class B, there is an object y in class A such that x is a part of y.

$$\forall A.\forall B.((Class(A) \wedge Class(B) \wedge Association(C) \wedge$$
$$memberEnd(C, b) \wedge type(b, B) \wedge aggregation(b, composite)\ )$$

$$\rightarrow \forall x.\ (B(x) \rightarrow \exists! y.(A(y) \wedge b(x,y)))$$

The following axioms are about enumeration classes.

Axiom 5. (Distinguishability of the literal constants)
The different literals in an enumeration class are different values.

$$\forall A.(Enumeration(A) \wedge ownedLiteral(A, v1) \wedge ownedLiteral(A, v2) \rightarrow (v1 \neq v2)) \quad \square$$

Axiom 6. (Completeness of the enumeration)
An enumeration class only has it literals as instances.

$$\forall A.(EnumClass(A) \rightarrow (\forall x.(A(x) \rightarrow \text{ownedLiteral(A, x)})) \quad \square$$

5.2. Axioms on Inheritance

The following axioms define the notion of inheritance.

Axiom 7. (Inheritance)
If class *A* inherits class *B*, every instance of *A* is also an instance of *B*.

$$\forall A\ .\forall B\ .(Class(A) \wedge Class(B) \wedge Inherits(A,B) \rightarrow \forall x\ (A(x) \rightarrow B(x))) \quad \square$$

Axiom 8. (Inherited attributes)
If class *A* inherits class *B*, every attribute of *B* is also an attribute of *A*.

$$\forall A.\forall B.(Class(A) \wedge Class(B) \wedge Inherits(A,B)$$
$$\rightarrow \forall x.(Property(x) \wedge HasAttribute(B, x) \rightarrow HasAttribute(A,x))) \quad \square$$

Axiom 9. (Inherited operations)
If class *A* inherits class *B*, every operation of *B* is also an operation of *A*

$$\forall A.\forall B.(Class(A) \wedge Class(B) \wedge Inherits(A,B)$$
$$\rightarrow \forall x.(Operation(x) \wedge HasOperation(B, x) \rightarrow HasOperation(A,x))) \quad \square$$

Axiom 10. (Abstract class)
If class *A* is abstract, for every object *x*, if *x* is an instance of class *A*, then, there must be a subclass *B* of *A* such that *x* is an instance of *B*.

$$\forall A.(\text{Class}(A) \wedge IsAbstract(A) \rightarrow \forall x.(A(x) \rightarrow \exists B.(Class(B) \wedge Inherits(B, A) \wedge B(x))) \quad \square$$

5.3. Axioms on Type Constraints

When classes are regarded as types, type consistency and type checking rule can be defined. This is reflected in the following axioms.

Axiom 11. (Attribute type)
If an attribute *a* of class *A* is of type class *B*, then, for all instance *x* of class *A*, the value of *x* on attribute *a* must be an instance of class *B*.

$$\forall A,B,a.(Class(A) \wedge HasAttribute(A, a) \wedge CurrentType(a, A, B)$$
$$\rightarrow (\forall x, y.\ (a(x, y) \wedge A(x) \rightarrow B(y)))) \quad \square$$

Axiom 12. (Association type constraint)
Let *a* be an association between classes *A* and *B*. For all objects *x* of class *A*, the objects *y* that *x* associates to through *a* must be in class *B*. Similarly, for all objects *y* of class *B*, the objects *x* that *y* associates to through *a* must be in class *A*.

$$\forall A.\forall B.(Class(A) \wedge Class(B) \wedge Association(a) \wedge memberEnd(a, Ea) \wedge$$
$$CurrentType(Ea, A) \wedge memberEnd(a, Eb) \wedge CurrentType(Eb, B)$$

$$\rightarrow (\forall x, y.\ Eb(x, y) \wedge A(x) \rightarrow B(y)) \wedge (\forall x, y.\ Ea(y, x) \wedge B(y) \rightarrow A(x))) \quad \square$$

The following axioms are about the redefinition of attributes and operations. Let $CurrentType(x, y, z)$ be a 3-ary predicate.

Axiom 13. (Redefined attributes)

If class A inherits class B and A declares an attribute a with type T_A, then the type of attribute a is T_A regardless what is defined in class B.

$$\forall A, B.(Class(A) \wedge Class(B) \wedge Inherits(A, B) \wedge OwnedAttribute(A, a) \wedge Type(a, T_A) \rightarrow CurrentType(a, A, T_A)) \quad \square$$

Axiom 14. (Unredefined attributes)

If class A inherits attribute a from B without redefining a, then the type of attribute a is as in B.

$$\forall A.\forall B.(Class(A) \wedge Class(B) \wedge Inherits(A, B) \wedge CurrentType(a, B, T_B) \wedge \neg OwnedAttribute(A, a) \rightarrow CurrentType(a, A, T_B)) \quad \square$$

Axiom 15. (Type of the literal constants)

The type of a literal value is the enumeration class in which the literal is declared.

$$\forall A.(EnumClass(\mathrm{A}) \wedge ownedLiteral(A, v) \rightarrow A(v)) \quad \square$$

5.4. Axioms on Multiplicity

The following axioms are about multiplicity.

Axiom 16. (Multiplicity of association)

Let a be an association between classes A and B. If the lower and upper limits of the multiplicity of a on class B's end are n and m, respectively, then for all objects x of class A, the number of objects associated to x through association a must between n and m.

$$\forall A.\forall B.(Class(A) \wedge Class(B) \wedge Association(a) \wedge memberEnd(a, Ea) \wedge type(Ea, A) \wedge memberEnd(a, Eb) \wedge type(Eb, B) \wedge upperValue(Eb, m) \wedge \quad lowerValue(Eb, n) \rightarrow (\forall x.\ A(x) \rightarrow n \leq || \{y \mid Eb(x, y)\} || \leq m)) \quad \square$$

Axiom 17. (Multiplicity of attributes)

Let a be an attribute of class A. If the multiplicity of attribute a has n and m as its lower and upper limits, then, for all objects x of class A, the number of objects as the value of x's the attribute a must between n and m.

$$\forall A.(Class(A) \wedge ownedAttribute(A, a) \wedge upperValue(a, m) \wedge lowerValue(a, n) \rightarrow (\forall x.\ A(x) \rightarrow n \leq || \{y \mid a(x, y)\ \} || \leq m)) \quad \square$$

Definition 9 (functional semantics of class diagrams)

Let M be any given class diagram in UML, the functional semantics of M consists of the axioms given above. We write $\square M \square_{Fun}$ to denote the functional semantics of M. $\square$

The above axioms hold for models at all layers of the metamodel hierarchy. This is the foundation for unifying the semantics of models and metamodels. The next section discusses how functional semantics can be integrated with the translation and hypothesis rules to further enhance the semantics for metamodels.

6. Integration of Functional and Descriptive Semantics

In this section, we discuss how functional semantics and descriptive semantics can be integrated into one logic system so that the semantics of metamodels can be formalized. We will first illustrate the way that two semantics are integrated then present a set of rules to derive formulas directly from class diagrams.

6.1. Integrating Two Semantics

Let's start with an example at model level. Consider the class diagram CD_1 depicted in Fig. 1. By applying the translation rules $TR1$ and $TR2$, we derive the following set of statements when the metamodel is B_1.

$$Class(Woman), \tag{1}$$

$$Class(Person), \tag{2}$$

$$Generalisation(wp), \quad specific(wp, Woman), \quad general(wp, Person). \tag{3}$$

These statements are descriptive and assert that class *Woman* inherits class *Person*. Formally,

$$Inherits(Woman, Person). \tag{4}$$

where the predicate *Inherits* is not derived from the metamodel using the signature mapping, but it is defined as follows using the predicates derivable from the metamodel.

$$Inherits(A,B) = \exists x\ (Generalisation(x) \wedge specific(x, A) \wedge general(x, B). \tag{5}$$

On the other hand, we have the following axiom (Axiom 7) in the functional semantics of objection orientation.

$$\forall A\ .\forall B\ .(Class(A) \wedge Class(B) \wedge Inherits(A,B) \rightarrow \forall x\ (A(x) \rightarrow B(x))) \tag{6}$$

Using formulas (1), (2) and (4), we derive the following statement from (6).

$$\forall x\ (Woman(x) \rightarrow Person(x)). \tag{7}$$

This statement is a property that objects of the system at run time must satisfy. It has been investigated in the research on semantics of UML, e.g. [10, 11].

Now, let's consider the metamodel of class diagram CD_2 depicted in Fig. 3. Applying the translation rules to this diagram, the following formulas can be obtained.

$$MetaClass(Class), \tag{8}$$

$$MetaClass(Classifier), \tag{9}$$

$$MetaGeneralisation(cc), \quad specific(cc, Class), \quad general(cc, Classifier). \tag{10}$$

where *MetaClass* is *Class* at meta-level and *MetaGeneralisation* is *Generalisation* at meta-level. They are introduced to avoid confusion. Thus, from (9) and the definition of *Inherits* in (5), we have that

$$Inherits(Class, Classifier). \tag{11}$$

This is again a descriptive statement about the metamodel CD_2. Since the axioms of functional semantics also apply to metamodel, we can derive the following statement from (8)~(11):

$$\forall x.(Class(x) \rightarrow Classifier(x)). \tag{12}$$

This is a statement that all models (i.e. the instances of metamodel CD_2) must satisfy.

From the above examples, we make two important observations.

First, the axioms of functional semantics are high order formulas, which contain variables that range over predicates; while formulas (7) and (12) are first order.

Second, and more importantly, formulas obtained by applying translation rules like (8) ~ (10) and formulas derived by applying the axioms are in different signatures. The predicate symbols derived from the meta-metamodel (e.g. *MetaClass* and *MetaGeneralisation*) are eliminated by the combining the functional and descriptive formulas. For example, formula (12) is in the same signature as formulas (1)~(3), rather than the signature of (8)~(10).

In general, for a model M at level i, the descriptive statements are in the signature derived from its metamodel N at level i+1. By combining them with the axioms of functional semantics, which contains predicate symbols derived from N and M, will generate statements in the signature derived from M, which is one level lower than N.

Consequently, by replacing functional axioms with formulas like (7) or (12), checking a system is an instance of a model can be done without looking at the metamodel. Similarly, checking if a model is an instance of a metamodel does not need to look at the meta-metamodel, etc. In other words, the instance-of relation is defined only involving two adjacent levels.

The following subsection demonstrates that formulas like (7) or (12) can be derived systematically without using a logic inference engine, but just a few transformation rules on the models.

6.2. Axiom Mapping

We now define a set of rules to derive formulas in first order logic from a class diagram. These rules are based on the functional semantics thus the formulas are the axioms to be satisfied by all its instances. Thus, the rules are called axiom mappings.

A. Classification of elements

There are two kinds of classes in a class diagram: concrete classes and abstract classes. Every element in an instance of a class diagram D must be an instance of at least one concrete class in D. Note that a M_1 model may depict only a subset of the classes in the modelled systems. Therefore the following axiom rule is applicable for a M_1 model under the hypothesis that all classes in the modelled system are depicted in the model. For a M_2 or M_3 model, however, the following axiom rule is always applicable because a M_2 or M_3 model must define all types of elements in its instance models. We have the following axiom rule to explicitly state the constraint.

Rule AR1. (Completeness of classification)

Let $\{C_1, C_2, \ldots C_n\}$ be the set of concrete classes in class diagram D. We include in $Axm(D)$ the formula.

$$\forall x.\ (C_1(x) \vee C_2(x) \ldots \vee C_n(x))\ .$$ □

For example, by applying rule $AR1$ on B_2 in Fig. 5, the following statement as the functional semantics of B_2 is derived, stating that the type of any element is *Generalisation*, *Class* or *Association*.

$$AR1(B_2) = \{\forall x.\ (Generalisation(x) \vee Class(x) \vee Association(x))\ \}$$

By applying $AR1$ on B_1 in Fig. 5, the following formula as the semantics of model B_1 is derived, which states that the type of any element is *Woman* or *Man*.

$$AR1(B_1) = \{\forall x.\ (Woman(x) \vee Man(x))\}$$

An element in a model has one and only one type; otherwise the element is incomprehensible. Therefore, if a model N is an instance of D, every element in N must be an instance of at most one concrete class in D. Hence the following axiom rule is defined.

B. Inheritance hierarchy

Inheritance hierarchy of classes represents taxonomy of concepts. "Each instance of the *specific classifier is also an indirect instance of the general classifier*" [12]. This relation can be expressed as logic implication between the predicates, thus we have the following axiom rule.

Rule AR2. (Logical implication of inheritance)

For a generalisation from class A to class B in a class diagram D, we include in $Axm(D)$ the following formula.

$$\forall x.\ (A(x) \rightarrow B(x))$$ □

For example, by applying $AR2$ to B_2 in Fig. 5, the following statements can be derived, stating that if an element is an instance of *Class* or *Association*, it is also an instance of *Classifier*.

$$AR2\ (B_2) = \{\forall x.\ (Class\ (x) \rightarrow Classifier(x)),\ \forall x.\ (Association\ (x) \rightarrow Classifier(x))\}$$

By applying $AR2$ to B_1 in Fig. 5, the following statements can be derived, stating that if an element is an instance of *Man* or *Woman*, it is also an instance of *Person.*

$$AR2\ (B_1) = \{\forall x.\ (Man(x) \rightarrow Person(x)),\ \forall x.\ (Woman(x) \rightarrow Person(x))\}$$

A model must have its elements completely and uniquely classified by classes in its metamodel. If model N is an instance of model D, an element in N as an instance of an abstract class in D must be an instance of some concrete class in D. Hence the following axiom rule is defined.

Rule AR3. (Completeness of specialisations)

Let A be an abstract class and C_1, C_2, …, C_k be the set of classes specialising A in a class diagram D. We include formula $\forall x.\ (A(x) \rightarrow (C_1(x) \vee C_2(x) \vee \ldots \vee C_k(x)))$ in $Axm(D)$. □

For example, by applying $AR3$to model B_2 in Fig. 5, the following statement can be

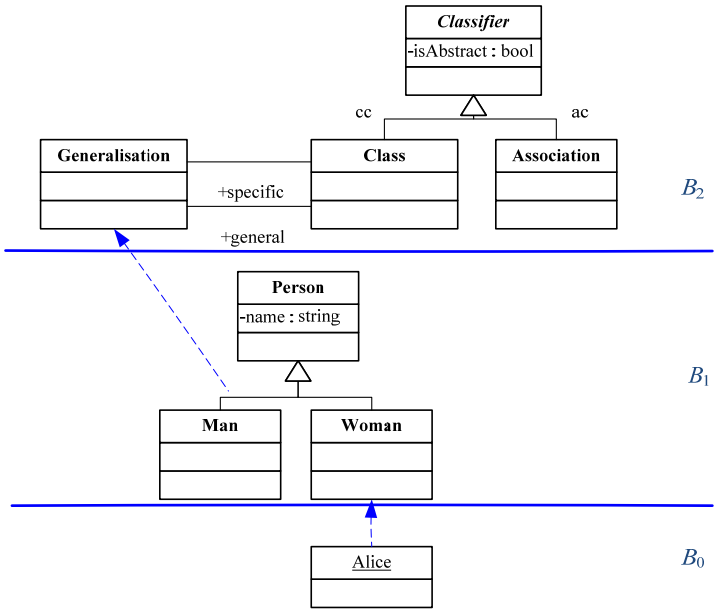


Fig. 5 Examples of models for applying axiom mapping rules.

derived, stating that if a model element is an instance of *Classifier*, its type must be either *Class* or *Association*.

$$AR3(B_2) = \{\forall x.\ (Classifier(x) \rightarrow (Class(x) \vee Association(x)))\}$$

By applying *AR*3to model B_1 in Fig. 5, the following statement can be derived, stating that if an element is an instance of *Person*, it must be an instance of either *Man* or *Woman*.

$$AR3\ (B_1) = \{\forall x.\ Person\ (x) \rightarrow Woman(x) \vee Man(x)\ \}$$

C. Type constraints

Binary predicates derived from associations and attributes in a model *D* define possible relationships between two elements in an instance of *D*. An axiom implicitly specified in *D* is such a relationship only exists between elements of certain types. Thus, we have the following axiom rule.

Rule AR4. (Type constraints)

For each binary predicate $A(x, y)$ derived from an association from metaclass C_1 to C_2 in *D*, or from an attribute *A* of type C_2 in a metaclass C_1, we include the following formula in *Axm*(*D*).

$$\forall x, y.\ (A(x, y) \wedge C_1(x) \rightarrow C_2(y))$$ □

For example, by applying *AR*4to B_2 in Fig. 5, the following statements are derived. The first, for instance, states that in an instance of B_2, the value of attribute *isAbstract* of a classifier must be a boolean value.

$$AR4(B_2) = \{\forall x, y.\ (isAbstract(x, y) \wedge Classifier\ (x) \rightarrow bool(y)),$$
$$\forall x, y.\ (specific(x, y) \wedge Generalisation\ (x) \rightarrow Class\ (y)),$$
$$\forall x, y.\ (general(x, y) \wedge Generalisation\ (x) \rightarrow Class\ (y))\}$$

By applying *AR*4to B_1 in Fig. 5, the following statement can be derived, stating that in an instance of B_1, the *name* of an object of *Person* must be a string.

$$AR4\ (B_1) = \{\forall x, y.\ (name(x, y) \wedge Person(x) \rightarrow string(y))\}.$$

D. Multiplicity constraints

Association ends and attributes are constrained by multiplicity. They "*constrains the size of the collection [...] of instances at the other end*"[12]. Thus, we have the following rule.

Rule AR5. (Multiplicity of binary predicate)

For each binary predicate $A(x, y)$ derived from an association from class C_1 to C_2 in *D*, let *Mul* be the multiplicity value specified on the association end *A*, we include formula below in *Axm*(*D*):

If *Mul* = 0..1: $\forall x, y, z.\ (C_1(x) \wedge A(x, y) \wedge A(x, z) \rightarrow (y = z))$

If *Mul* = 1..*: $\forall x.(\ C_1(x) \rightarrow \exists\ y.\ A(x, y))$

If *Mul* = 2..*: $\forall\ x.(\ C_1(x) \rightarrow \exists\ y, z.\ A(x, y) \wedge A(x, z) \wedge (y \neq z))$

If *Mul* = 1 or unspecified:

$$\forall x.\ (C_1(x) \rightarrow \exists\ y.\ A(x, y)),\ \forall x, y, z.\ (C_1(x) \wedge A(x, y) \wedge A(x, z) \rightarrow (y = z))$$

If *Mul* = 0..2:

$$\forall x, y, z, u.(\ C_1(x) \wedge A(x, y) \wedge A(x, z) \wedge A(x, u) \rightarrow (y = z) \vee (y = u) \vee (u = z))$$ □

For example, by applying AR5to B_2 in Fig. 5, the following statements are derived, stating that any generalisation element in an instance of B_2 must have a single specific end

and a single general end.

$$\text{AR5}\ (B_2) = \{\forall x.\ (Generalisation(x) \rightarrow \exists y.\ specific(x, y)),$$
$$\forall x, y, z.\ (Generalisation\ (x) \wedge specific\ (x, y) \wedge specific\ (x, z) \rightarrow (y = z)),$$
$$\forall x.\ (Generalisation(x) \rightarrow \exists y.\ general(x, y)),$$
$$\forall x, y, z.\ (Generalisation\ (x) \wedge general\ (x, y) \wedge general\ (x, z) \rightarrow (y = z))\ \}$$

E. Properties of enumeration values

Each enumeration class in a model defines a data type, and the enumeration values defined in the enumeration class are the domain of the data type. With signature mapping, we can map enumeration values into constants. The following axiom rules explicitly state that the constants are instances of the enumeration class, are distinguishable from one another, and define a complete domain of the data type.

Rule AR6 (Distinguishability of the literal constants):

For each pair of different literal values a and b of an enumeration type, we include a formula $a \neq b$ in $Axm(D)$. □

Rule AR8 (Type of the literal constants):

For each enumeration value a defined in an enumeration class E, we include the formula $E(a)$ in $Axm(D)$. □

Rule AR9 (Completeness of the enumeration):

An enumeration type only contains the listed literal constants as its values, hence for each enumeration class E with literal values $a_1, a_2, \ldots, a_k$, we include the following formula in $Axm(D)$.

$$\forall x.\ E(x) \rightarrow (x = a_1) \vee (x = a_2) \vee \ldots \vee (x = a_k)$$ □

For example, by applying AR_7 to A_3 in Fig. 4, the following axioms on instances of A_3 are derived, stating that constants *none*, *shared* and *composite* are different values.

$$AR_7(A_3) = \{none \neq shared,\ none \neq composite,\ composite \neq shared\}$$

By applying AR_8 to A_3, the following axioms on instances of A_3 are derived, stating that constants *none*, *shared* and *composite* have *AggregationKind* as their type.

$$AR_8(A_3) = \{AggregationKind(none),$$
$$AggregationKind(shared),$$
$$AggregationKind(composite)\}$$

By applying AR_9 to A_3, the following axioms on instances of A_3 are derived, stating that constants *none*, *shared* and *composite* are the complete set of values of type *AggregationKind*.

$$AR_9(A_3) = \{\forall x.\ AggregationKind\ (x) \rightarrow (x = none) \vee (x = shared) \vee (x = composite)\ \}$$

The following theorem proves that the axiom mapping rules are correct with respect to the functional axioms.

Theorem 2 (Correctness and Completeness of Functional Semantics Mapping)

For all class diagrams D *and its metamodel* N, for all formulas ϕ in $Axm(D)$, we have that $\square D \square_{Des} \cup \square N \square_{Fun} \vdash \phi$.

Proof. (sketch)

We prove the statement by proving that the formulas generated by each axiom mapping can be derived from the axioms of functional semantics and the formulas in the descriptive

semantics. Let ϕ in $Axm(D)$.

Case 1: When ϕ is generated by applying Rule AR1, we have that

$$\phi = \forall x.\ (C_1(x) \vee C_2(x) \ldots \vee C_n(x))$$

where, according to the condition of Rule AR1, $\{C_1, C_2, \ldots C_n\}$ is the set of concrete classes in class diagram D. Thus, according to Rule TR1, we have that $Class(C_1)$, $Class(C_2)$, …, $Class(C_n)$ in $\Box D\Box_{Des}$. By Axiom 1, we have that

$$\forall x\ .(\exists C.(Class(C) \wedge C(x))).$$

Let x be any given object. Assume that C is the class such that $Class(C) \wedge C(x))$ holds. If C is a concrete class, C is one of $C_1, C_2, \ldots, C_n$. Thus, the statement is true. If C is an abstract class, by TR2, we have that $IsAbstract(C, True) \in \Box D\Box_{Des.}$ Then, by Axiom 10, we have that there is C' in D such that $Class(C')$, $Inherits(C',C)$ and $C'(x)$. If C' is a concrete class, then, the statement is true; otherwise, repeat the above argument for a finite number of times, we can find a concrete class C'' such that $C''(x)$ is true. Since $\{C_1, C_2, \ldots C_n\}$ contains all concrete classes in class diagram D, we can deduce that $C'' \in \{C_1, C_2, \ldots C_n\}$. Therefore, $\forall x.\ (C_1(x) \vee C_2(x) \ldots \vee C_n(x))$ is true. In other words, $\Box D\Box_{Des} \cup \Box N\Box_{Fun} \vdash \phi$.

Case 2: When ϕ is generated by applying Rule AR2, we have that

$$\phi = \forall x.\ (A(x) \rightarrow B(x))$$

where, according to the condition of Rule AR2, we have that A and B are classes and there is a generalisation from class A to class B in a class diagram D. Thus, according to Rule TR1, we have that Class(A), Class(B), and Inherits(A,B) are in $\Box D\Box_{Des}$. Therefore, by Axiom 7, we deduce that $\forall x.\ (A(x) \rightarrow B(x))$. That is, $\Box D\Box_{Des} \cup \Box N\Box_{Fun} \vdash \phi$.

Case 3: When ϕ is generated by applying Rule AR3, we have that

$$\phi = \forall x.\ (A(x) \rightarrow (C_1(x) \vee C_2(x) \vee \ldots \vee C_k(x))),$$

where, A is an abstract class and $\{C_1, C_2, \ldots, C_k\}$ is the set of classes specialising A in a class diagram D. Therefore, we have that the following formulas are in $\Box D\Box_{Des}$.

$$Class(\mathrm{A}), IsAbstract(A, True), Class(C_1), \ldots, Class(C_k).$$

By Axiom 10, we have that, for all x such that $A(x)$ is true, there is a class B such that B is a subclass of A and $B(x)$ is true. Since $\{C_1, C_2, \ldots, C_k\}$ contains all subclasses of A, we have that $B \in \{C_1, C_2, \ldots, C_k\}$. In other words, $(C_1(x) \vee C_2(x) \vee \ldots \vee C_k(x))$ is true. Therefore, we can deduce that $\forall x.\ (A(x) \rightarrow (C_1(x) \vee C_2(x) \vee \ldots \vee C_k(x)))$. That is, $\Box D\Box_{Des} \cup \Box N\Box_{Fun} \vdash \phi$.

Case 4: When ϕ is generated by applying Rule AR4, we have that

$$\phi = \forall x, y.\ (A(x, y) \wedge C_1(x) \rightarrow C_2(y))$$

where binary predicate $A(x, y)$ either represents an association from metaclass C_1 to C_2 in D, or represents an attribute A of type C_2 in a metaclass C_1. Here, we only give the proof for the case when the predicate A represents an attribute in metaclass C_1. The other case is similar, hence omitted for the sake of space.

When the predicate A represents an attribute in metaclass C_1 and attribute A's type is C_2, according to TR1 and TR2, we have that formulas $Class(C_1)$, $HasAttribute(C_1,A)$ are in $\Box D\Box_{Des}$ and formula $CurrentType(A,C_1,C_2)$ can be deduced from the definition of predicate $CurrentType$ and formulas in $\Box D\Box_{Des}$. By Axiom 11 below,

$$\forall A.(Class(A) \wedge HasAttribute(A, a) \wedge CurrentType(a, A, B)$$
$$\rightarrow (\forall x, y.\ (a(x, y) \wedge A(x) \rightarrow B(y)))),$$

we have that $(\forall x, y.\ (A(x, y) \wedge C_1(x) \rightarrow C_2(y)))$. That is, $\Box D\Box_{Des} \cup \Box N\Box_{Fun} \vdash \phi$.

Case 5: When ϕ is generated by applying Rule AR5, the proof is very similar to the proof given in Case 4, but the axioms in the functional semantics used in the proof are Axiom 16

and 17. Details are omitted for the sake of space.

Case 6: When ϕ is generated by applying Rule AR6, the proof is very similar to the proof given in Case 3, but the axioms in the functional semantics used in the proof are Axiom 5. Details are omitted for the sake of space.

Case 7: When ϕ is generated by applying Rule AR7, the proof is very similar to the proof given in Case 3, but the axioms in the functional semantics used in the proof are Axiom 15. Details are omitted for the sake of space.

Case 8: When ϕ is generated by applying Rule AR8, the proof is very similar to the proof given in Case 1, but the axioms in the functional semantics used in the proof are Axiom 6. Details are omitted for the sake of space.

□

Note that, the above proof of the correctness of the axiom rules also demonstrate that the rules cover all axioms except Axiom 2 and 3, which can be regarded as 'definitions' of the predicates *HasOperation* and *HasAttribute*. This suggests that this set of rules is complete with regard to the set of axioms of functional semantics in the sense that, for all first order formulas ϕ in the signature of $Sig(N)$, $\Box D\Box_{Des} \cup \Box N\Box_{Fun} \vdash \phi$ implies that $\Box D\Box_{Des} \cup Axm(D) \vdash \phi$. Intuitively, for each axiom of the functional semantics, there is a corresponding axiom rule. Therefore, a deduction of a formula ϕ from $\Box D\Box_{Des} \cup \Box N\Box_{Fun}$ can be replaced by an equivalent deduction of the formulas ϕ from $\Box D\Box_{Des} \cup Axm(D)$ as far as ϕ does not contain higher order variables. However, the rigorous proof of the completeness still remains open, and will be a topic for future work.

Also note that UML class diagram can be complemented with constraints in OCL. For example, well-formedness rules as a part of the UML metamodel are specified in the UML documentation [12]. Such OCL constraints are also axioms that instances of a class diagram must satisfy. Thus, we have an additional rule that does not correspond to any axioms of functional semantics.

Rule AR10 (OCL constraints).

For each constraint formally specified in OCL, we include a corresponding formula in $Axm(D)$. □

Note that this rule is also applicable to class diagrams used as models in all layers of the UML metamodel hierarchy.

7. Applications of the Formal Semantics

In this section, we discuss the application of the formal semantics in model-driven software development.

7.1. Applications of descriptive semantics

In Section 5, we illustrated with examples that the descriptive semantics mapping can be applied to models at any layer in the multi-layer hierarchy. In the view that 'a model is a set of statements in some modelling language'[13], descriptive semantics of a model represents the model's statements in a first order logic which is derived from the modelling language by applying signature mapping on the metamodel of the language. Predicates in the first order logic represent element types in the model and relationships between the elements, regardless of how to interpret the element types and the relationships in a subject domain. As shown in our previous work [4], descriptive semantics mapping is applicable not only to class diagrams, but also to any other types of diagrams in M_1 models. As long as the *type* of each element and the *relationship* between elements can be identified, descriptive semantics

mapping is applicable. This is the reason why descriptive semantics mapping can be applied on models at different layers in the metamodel hierarchy.

Descriptive semantics has several applications. First, based on the definition of 'instance of', descriptive semantics can be used to reason if a model satisfies a metamodel. We have conducted case studies on some UML models to check their well-formedness [4]. The models were translated by LAMBDES into logic systems in SPASS format and their logic properties were verified using SPASS. For M_2 models, the descriptive semantics provides a way to logically prove if it is a valid instance of the M_3 model.

Second, when a subject domain is regarded as a collection of mathematical structures, the descriptive semantics of a model can be evaluated to a truth value with respect to a structure in the subject domain. Therefore, descriptive semantics of a model can be used to evaluate if a model is satisfied by a system. Hypothesis mapping explicitly represent the specific use of the model, therefore provides the flexibility of interpreting models differently in different context.

Third, as descriptive semantics are logical representations of the content of a model, it can be used to reason about certain properties of the model. In our previous work, the descriptive semantics of M_1 models has been used to analyse their consistency with respect to user-defined consistency rules [4]. Descriptive semantics of UML class diagrams has also been used to recognise patterns from software designs, and to formally analyse the logic relations between design patterns [1, 14].

7.2. Applications of functional semantics

In section 4, we illustrated with examples that functional semantics mapping can be applied to models at any layer in the multi-layer hierarchy. Functional semantics formalises the properties of basic OO concepts through the mappings defined on UML class diagrams, because the constructs in class diagrams represent the OO concepts. Since M_2 and M_3 models are all UML class diagrams and based on the same OO conception, functional semantics mapping can be equally applied to them.

For M_2 models, functional semantics can be used to verify if a class diagram is a well-defined metamodel of some models. We conducted case studies on the functional semantics of UML 2.0 metamodel and a profile for AspectJ [7]. Using our prototype tool LAMBDES, the metamodels were translated into logic systems in the SPASS format and their logic properties such as consistency and completeness were checked by invoking SPASS. Inconsistencies and incompleteness were discovered in the metamodels.

For M_1 models, functional semantics provides a way to generate properties of programs from models. Properties described in a UML model, such as multiplicity specifications or OCL rules, are a part of the functional semantics of the model, and hence axioms over the run-time behaviour of the modelled system. Such constraints can be used to formally verify a program, or automatically inserted into programs as assertions or pre/post-conditions for during the code-generation phase of MDE.

8. Related work

With UML gaining popularity of in the past two decades, great efforts have been made to formalise the semantics of UML models and metamodels, e.g. [15, 16]. The most closely related works are those addressing the semantics of basic concepts of the metamodel hierarchy, such as models, interpretation of models, metamodels and conformance of models to metamodels [17, 18]. Among them, Poernomo [19] formalises the metamodels

and the conformance of models to a metamodel based on type lambda calculus. Boronat and Meseguer [4], and Egea and Rusudefines [18] define the semantics of MOF in membership equational logic (MEL).

The following compares our approach with the existing work by discussing how key issues in the formalisation of UML metamodel hierarchy were addressed differently.

8.1. On the metamodel hierarchy

It is recognised that many artefacts, besides UML models, can also be considered as models and the languages specifying them as metamodels in the four-layer metamodel hierarchy [18-20]. Examples of M_0, M_1 and M_2 models in different technical spaces are:

- XML: documents, schemas and the schemas of XML Schema;
- EBNF: programs, grammars and the grammar of EBNF;
- DBMS: instantiated database tables, database table declarations and database model.

Viewing them in a same layered metamodel hierarchy enables to tackle the problems on the coordination between the artefacts and the interoperability of their supporting tools, which is an important topic in the context of MDE. Existing techniques for transforming models include XMI (XML Metadata Interchange) for bridging with the XML space, JMI (Java Metadata Interchange) for bridging with the Java space, CMI (Corba Model Interchange) for bridging with the Corba space, etc. Bézivin et al [20] pointed out that such techniques are under the principle of metamodel-driven model transformations in the sense that transformations are developed according to M_2 layer so as to transform models at M_1 layer.

Incorporating artefacts from various technical spaces in a same layered metamodel hierarchy, on one hand, reveals that a same real-world thing can be captured by different artefacts. On the other hand, when a UML-centric viewpoint is taken, it enables to explain the semantics of UML models within various technical spaces. In this paper, we examine the *logic* relations between these artefacts and regard the artefacts as forming subject domain of UML models. The formal definition of subject domain characterises the widely used intuitive notion of the system being modelled. Consequently, UML models can be interpreted to many other structures beyond software systems or systems in the real world. To our knowledge, none of existing researchers take this view on the interpretation of models.

8.2. On semantics of models

Addressing the under-specification and ambiguity in UML's semantics, remarkable efforts have been made in the past decade to formalise UML semantics. Much of the publications are about the functional semantics aiming at 'a deeper understanding of OO [13].The following proposals are among the most well-known.

The formalisation of class diagram is considered the most important type of diagrams in UML, and a number of proposals have been advanced. Evans *et al*. have used Z schemas to define classifier, association, generalisation and attribute etc.[20]. Relations between objects and classifiers are specified as axioms. Diagrammatical transformation rules are defined as deduction rules to prove properties of UML models. There are a number of other researchers who have also used Z or its variants, such as Object-Z, to formalising class diagram; see [21] for a survey of different approaches of this type. First order logic (FOL) and description logics (DLs) have been used to formalise class diagram, too [10]. By encoding UML class diagrams in DL knowledge bases, DL reasoning systems can be used to reason about class

diagrams. Our work on the functional semantics is inspired in the works on logic representations of class diagrams. However, we differ from others by specifying the axioms in higher predicate order logic in the signature derived from the metamodels. Therefore, our definition of the functional semantics is independent of the model and the layer on which the model is interpreted. Our rules that derive the functional semantics of a particular model are formally proved to be correct with respect to the axioms.

Formalisation of other types of diagrams has also been investigated, especially on state machine diagram. For example, Varro [22] has proposed a rule-based operational semantics of state machine based on transition systems. Another work on operational semantics of state machine has been reported in [23]. Great efforts have also been made on formalising different diagrams in one semantic framework. Considering the semantics of a UML model as a set of acceptable structured process, Reggio, Cerioli and Astesianothe [24] map class diagrams and state machines into algebraic specifications in Casl-ltl. Kuske et al. has employed graph transformation in an attempt to integrate semantics of class diagram, object diagram and state machine diagrams [25]. In our previous work on the formalisation of UML, we have also formalised of other types of UML diagrams such as sequence diagram and state machines in a unified framework, which is generalised in this paper. Readers are referred to [4] for details. Comparison with related works in this direction is beyond the scope of this paper, thus omitted.

To bridge the gap between UML and formal methods, the extensibility mechanism of UML *profile* is used to define specialisations of UML. In [26], a profile UML-B is designed so that the semantics of specialised UML entities is defined via a translation into B. In [27], Moller et al. used a combination of the process algebra CSP and the specification language Object-Z as the intermediate specification language to link UML and Java. A UML profile for CSP-OZ is designed with the aim of generating part of the CSP-OZ specifications from the specialised UML models.

The above existing methods define the semantics of UML by mapping models into a specific semantic domain, such as labelled transition systems, or OO software systems specified in a formal notation such as Z. The properties of OO systems are specified as axioms and used to reason about UML models. In other words, they mostly addressed the functional semantics of UML. Each method focuses on certain properties of OO systems, hence a certain subset of UML is formalised. However, it is hard to see how these approaches could work either alone or together for the full-fledged UML. Most importantly, the ambiguity in descriptive semantics is not addressed in these works. Instead, their semantics formalisations are based on explicit or implicit assumption on the descriptive semantics. Automation of translating UML models to formal specifications to facilitate automated reasoning of UML models has not been achieved in the existing methods.

As a recent effort towards the executable semantics of UML, OMG launched the Semantics of a Foundational Subset for Executable UML Models (fUML) [13]. On the introduction section on the semantics of models, it is stated that '*the same model may have different "meanings" under different interpretations*'. On the semantics of metamodels, fUML also regards '*the statements of the metamodel as axioms about the modelling language*'. However, such notions are only informally explained through examples, but not reflected in the semantics definitions.

In our approach, we make it explicit how models can be interpreted differently in different usage context through hypothesis mappings. In this paper, we discussed the particular usage of class diagrams as metamodels and presented a set of hypothesis mapping rules to derive the formulas represent such hypothesis from class diagrams.

8.3. On semantics of metamodels

The formal definition of modelling language BON reported in [28] is similar to our approach. In [28], the metamodel of BON is depicted in BON notation and then specified in formal specification language PVS. Modelling concepts of BON, including *abstractions* such as Class and Feature and *relationships* such as Aggregation and Association, are specified as types in PVS. Inheritance hierarchy in the metamodel are mimicked by subtype relations. The semantic relations between the modelling concepts are defined as functions in PVS. The signature of a PVS system is manually defined according to the metamodel. Then, well-formedness constraints on BON models are specified as axioms in PVS. When BON models are formalised in PVS, their well-formedness with respect to the metamodel can be checked using PVS theorem prover. It is reported that the BON metamodel was analysed and debugged through the formalisation. In comparison, we view a metamodel as more than the definition of the signature of the modelling language. For example, from an inheritance hierarchy in a metamodel, not only types of model elements and subtype relations can be generated, but also axioms on the classification of model elements. Moreover, our method is applicable to all metamodels. In other words, the domain of the semantics mapping is the set of metamodels in UML class diagrams rather than a specific metamodel for a specific language.

Viewing the role of a metamodel in the four-layer metamodel hierarchy as a type of models, a few proposals on the semantics of metamodels and MOF have been reported in the literature [19, 29].

Similar to our distinction of descriptive semantics and functional semantics, Poernomo identify two aspects of a metamodel: as an object-based representation (as data) and as a class-based representation (as a type of models) [19]. A higher-order typed lambda calculus with dependent sum and product types in Constructive Type Theory (CTT) is used to formalise the semantics of metamodels. Classes and objects are treated using recursive records. The four levels of the MOF correspond to the CTT's predicative hierarchy of type universes, where Type_0, Type_1, Type_2, . . . are defined. M_2 level classifiers, for instance, are given a dual representation as objects of the MOF class types and as Type_1 class types. In this framework, the conformance relation is implicitly provided by construction: only valid models can be defined as terms, and their definition constitutes a formal proof of the fact that the model belongs to the corresponding type by means of the Curry-Howard isomorphism.

Boronat and Meseguer propose an algebraic semantics for MOF [29]. The problems they address are similar to ours, i.e., the basic notions of the hierarchy not yet fully formally defined in the current MOF standard, including what is a model, what is a metamodel and what is reflection in the MOF framework, etc. They present a reflective, algebraic, executable framework for precise metamodelling based on membership equational logic (MEL) that supports the MOF standard. The formal framework provides a formal semantics of the basic notions. In particular, they formalize the notions of: (i) model type which is a type in MEL allowing models to be considered as first-class citizens, (ii) metamodel realization which is a MEL theory referring to the mathematical representation of a metamodel, and (iii) conformance relation, by means of a reflective semantics that associates a mathematical metamodel realization to each metamodel in MOF. By using the Maude language, which directly supports MEL specifications, this formal semantics is executable. This executable semantics has been integrated within the Eclipse Modeling Framework as a plug-in tool called MOMENT2.

Egea and Rusu investigate conformance of models to metamodels by formalising models

with MEL [30]. First, two invariants are defined: a metamodel does not have cyclic generalizations and each association is linked to two classes. Then, both metamodels enriched with OCL invariants and models are represented as MEL specifications. Two levels of conformance are defined: structurally conformant and semantical conformance. A model is structural conformance to a metamodel if the model theory provides an actual interpretation of the MEL specification denoting the metamodel. Semantical conformance requires, in addition to structural conformance, that all the invariants imposed on the metamodel become true in its instance model.

In comparison, Egea and Rusu's notion of structural conformance is similar to ours, and their way to evaluate the conformance of a model to a metamodel through formalising them to logic theories is also close to ours. There are two key differences. First, we regard OCL invariants within a model as part of the model. With axiom mapping, OCL invariants within a model are syntactically transformed to first order formulas, which is a part of the axioms imposed on the instances of the model. Second, the two pre-defined invariants are unnecessary in our semantics definition. In particular, if there is a generalization cycle in a class diagram, axioms generated from it are logically inconsistent, indicating that it cannot acts as a metamodel of some models. A formal definition of well-defined metamodel was given in [7]. This difference reveals that our axiom mapping is sufficient and necessary to express the properties that a well-defined metamodel must hold . We do not require an association to have two ends, as it is stated in UML superstructure that '*An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end.*' [12] In summary, we do not impose additional information to the semantics of a metamodel.

9. Conclusion

In this section, we summarise the contribution of this paper and discuss further work.

9.1. Summary

The contribution of the paper is a unified semantic framework for the multi-layer metamodel hierarchy. For an individual model, its descriptive semantics and functional semantics are distinguished to capture different aspects of semantics of models. They are integrated by linking the functional semantics of metamodel at layer M_{i+1} to the descriptive semantics of models at layer M_i, where i can be any natural number ≥ 0.

Our semantics provides clear and formal definitions of the basic concepts in the metamodel hierarchy. First, the semantics of a model is a set of statements about the system under study. These statements are represented as predicate logic formulas in the signature defined by the metamodel of the model. Furthermore, they are classified into two parts, descriptive ones and functional ones. The former is used to judge if a system is an instance of the model, and the latter is about the properties of the functionality and dynamic behaviours of the system.

Second, the concept of subject domain of a model is formally defined to be a set of mathematical structures of the signature defined by the model. Therefore, whether a collection of structures qualify to be a subject domain of some models can be precisely determined. Not only systems can be regarded as such mathematical structures, but also models at various layers.

Third, the instance-of relationship between system and model (also between models and metamodels) is formally defined, which enables to precisely determine the relationship

between models and metamodels through logic reasoning. The semantic definition is equally applicable to various layers in the metamodel hierarchy.

Finally, we revised the semantics mapping rules that we proposed in our previous work on UML models so that they are applicable to all layers. In this paper, we also proved the correctness and completeness of the axiom mapping rules with respect to the static functional semantics. We have also proved the correctness of descriptive semantics mapping rules and the correctness of employing a theorem prover to validate the instance-of relation between models and metamodels.

9.2. Future work

We have considered essential elements in class diagrams in the current semantics definition, but have not considered some elements e.g. visibility property. To express the semantics, especially the functional semantics of such elements, is among our further work.

We will also explore the application of the semantics definitions to various model analysis tasks in MDE. One possible direction is to apply functional semantics on M_1 model for model-driven program verification. Functional semantics of M_3 model can be used to verify the logic consistency of meta-metamodel as well as the well-formedness of M_2 models. We will also investigate the mechanism of reflection in MOF model.

The aim of the four-layer metamodel hierarchy is to facilitate the interchange of models in different formats. To bridge different technical spaces, research on model transformations and tool interoperability based on metamodels and meta-metamodels have been reported in the literature [20, 31, 32]. We consider to work on this direction based on the semantics presented in this paper and applying institution theory [33] in the similar way that graphic extension of BNF is studied [34].

References

[1] Zhu H, Shan L, Bayley I and Amphlett R. A formal descriptive semantics of UML and its applications. In: UML 2 Semantics and Applications, Lano K, eds. 2009, John Wiley & Sons, Inc. 95-123.

[2] OMG. Unified Modeling Language: Infrastructure. Version 2.3. Object Management Group. 2010. http://www.omg.org/spec/UML/2.3/. (Last access: Feb 2011)

[3] OMG. Meta Object Facility (MOF) 2.0 Core Specification. Object Management Group. 2006. http://www.omg.org/cgi-bin/doc?formal/2006-01-01. (Last access: Feb 2011)

[4] Shan L and Zhu H. A formal descriptive semantics of UML. In: Proc. of the 10th International Conference on Formal Engineering Methods (ICFEM 2008). 2008. 375-396.

[5] Seidewitz E, What models mean. IEEE Software, 2003. **20**(5): 26 - 31.

[6] Max Planck Institut Informatik. SPASS: An Automated Theorem Prover for First-Order Logic with Equality. 2011. http://www.spass-prover.org. (Last access: Jan 2011)

[7] Shan L and Zhu H. Semantics of metamodels in UML. In: Proc. of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009). 2009. 55-62.

[8] Dong J, Zhao Y and Peng T. Architecture and design pattern discovery techniques – a review. In: Proc. of the 2007 International Conference on Software Engineering Research and Practice (SERP 2007). Volume II. 2007. 621-627.

[9] Atkinson C. Meta-modeling for distributed object environments. In: Proc. of the 1st International Conference on Enterprise Distributed Object Computing. 1997. 90-101.

[10] Berardi D, Cal A and Calvanese D. Reasoning on UML class diagrams. Artificial Intelligence, 2005. **168**(1): 70 - 118.

[11] Kaneiwa K and Satoh K. Consistency checking algorithms for restricted UML class diagrams. In: Proc. of the 4th International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006). 2006. 219 - 239.

[12] OMG. Unified Modeling Language: Superstructure. Version 2.3. Object Management Group. 2010. http://www.omg.org/spec/UML/2.3/. (Last access: Feb 2011)

[13] OMG. Semantics of a Foundational Subset for Executable UML Models. 1.0. Beta 3 Edition. 2010. http://www.omg.org/spec/FUML/. (Last access: Feb 2011)

[14] Zhu H, Bayley I, Shan L and Amphlett R. Tool support for design pattern recognition at model level. In: Proc. of the 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009). 2009. 228-233.

[15] Clark T, Evans A and Kent S. The metamodelling language calculus: Foundation semantics for UML. In: Proc. of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001). 2001. 17-31.

[16] Favre L. Foundations for MDA-based forward engineering. Journal of Object Technology, 2005. **4**(1): 129-153.

[17] Bézivin J. On the unification power of models. Software and System Modeling, 2005. **4**(2): 171-188.

[18] Kühne T. Matters of (meta-) modeling. Software and Systems Modeling, 2006. **5**(4): 369-385.

[19] Poernomo I. The meta-object facility typed. In: Proc. of the 21st Annual ACM Symposium on Applied Computing (SAC 2006). 2006. 1845-1849.

[20] Bézivin J, Devedzic V, Djuric D, Favreau J-M, Gasevic D and Jouault F. An M3-neutral infrastructure for bridging model engineering and ontology engineering. In: Proc. of the 1st International Conference on Interoperability of Enterprise Software and Applications. 2005. 159-171.

[21] Amálio N and Polack F. Comparison of formalisation approaches of UML class constructs in Z and Object-Z. In: Proc. of the 3rd International Conference of B and Z Users (ZB 2003). LNCS 2651. 2003. 339 - 358.

[22] Varro D. A formal semantics of UML statecharts by model transition systems. In: Proc. of the 1st International Conference on Graph Transformation (ICGT 2002). LNCS 2505. 2002. 378 - 392.

[23] Beeck Mvd. A structured operational semantics for UML-statecharts. Software and System Modeling, 2002. **1**(2): 130 - 141.

[24] Reggio G, Cerioli M and Astesiano E. Towards a rigorous semantics of UML supporting its multiview approach. In: Proc. of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001). LNCS 2029. 2001. 171 - 186.

[25] Kuske S, Gogolla M, Kollmann R and Kreowski H-J. An integrated semantics for UML class, object and state diagrams based on graph transformation. In: Proc. of the 3rd International Conference on Integrated Formal Methods (IFM 2002). LNCS 2335. 2002. 11 – 28.

[26] Snook C and Butler M. UML-B: Formal modeling and design aided by UML. ACM Transactions on Software Engineering and Methodology, 2006. **15**(1): 92-122.

[27] Möller M, Olderog E-R, Rasch H and Wehrheim H. Linking CSP-OZ with UML and Java: A case study. In: Proc. of the 4th International Conference on Integrated Formal Methods (IFM 2004). 2004. 267 - 286.

[28] Paige RF and Ostroff JS. Metamodelling and conformance checking with PVS. In: Proc. of the 4th International Conference on Fundamental Approaches to Software Engineering (FASE 2001). 2001. 2 - 16.

[29] Boronat A and Meseguer J. An algebraic semantics for MOF. Formal Aspects of

Computing, 2010. **22**(3-4): 269-296.

[30] Egea M and Rusu V. Formal executable semantics for conformance in the MDE framework. Innovations in Systems and Software Engineering, 2010. **6**: 73–81.

[31] Bruneliere H, Cabot J, Clasen C, Jouault F and Bézivin J. Towards model driven tool interoperability: Bridging Eclipse and Microsoft modeling tools. In: Proc. of the 6th European Conference on Modelling Foundations and Applications (ECMFA 2010). 2010. 32-47.

[32] Jouault F, Vanhooff B, Bruneliere H, Doux G, Berbers Y and Bézivin J. Inter-DSL coordination support by combining metamodeling and model weaving. In: Proc. of the 25th Annual ACM Symposium on Applied Computing (SAC 2010). 2010. 2011-2018.

[33] Goguen JA. Data, schema, ontology and logic integration. Logic Journal of the IGPL, 2005. **13**(6): 685-715.

[34] Zhu H. On the theoretical foundation of meta-modelling in graphically extended bnf and first order logic. In Proc. of the 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2010). 2010. 95-104.