Formalizing Domain-Specific Metamodeling Language XMML Based on First-order Logic

Tao Jiang
School of Mathematics and Computer Science, Yunnan University of Nationalities, Kunming, P.R.China
Email: jtzwy123@gmail.com

Xin Wang
School of Mathematics and Computer Science, Yunnan University of Nationalities, Kunming, P.R.China
Email: wxkmyn@yahoo.com.cn

Abstract—**Domain-Specific Modeling has been widely and successfully used in software system modeling of specific domains. In spite of its general important, due to its informal definition, Domain-Specific Metamodeling Language (DSMML) cannot strictly represent its structural semantics, so its properties such as consistency cannot be systematically verified. In response, the paper proposes a formal representation of the structural semantics of DSMML named XMML based on first-order logic. Firstly, XMML is introduced, secondly, we illustrate our approach by formalization of attachment relationship and refinement relationship and typed constraints of XMML based on first-order logic, based on this, the approach of consistency verification of XMML itself and metamodels built based on XMML is presented, finally, the formalization automatic mapping engine for metamodels is introduced to show the application of formalization of XMML.**

Index Terms—**Domain-Specific Metamodeling Language, structural semantics, attachment, refinement, consistency verification**

I. INTRODUCTION

Compared with the uniformity and standardization of MDA [1], DSM [2] focuses on simplicity, practicability and flexibility. As a metamodeling language for DSM, DSMML plays an important role in system modeling of specific areas.

DSMML is a metalanguage used to build Domain-Specific Modeling Languages (DSMLs); this process that we use DSMML to build domain metamodels indicating the structural semantics of DSMLs is called metamodeling. Correspondingly, DSML is modeling language used to build domain application models; this process that we use DSML to build domain application models is called application modeling.

Semantics of DSMML can be grouped into structural semantics [3] and behavioral semantics. The former concerns static semantic constraints of relationship between modeling elements, focusing on the static structural properties; the latter concerns execution semantics of domain metamodels, focusing on the dynamic behavior of the metamodels. Although structural semantics is very important, research in structural semantics is not as extensive and deep as behavioral semantics', so this paper only studies structural semantics of DSMML.

There are several problems that have not been solved well for DSMML, which include precise formal description of its semantics, method of verification of properties of domain metamodels based on formalization and automatic translation from metamodels to corresponding formal semantic domain.

The paper proposes a formal representation of the structural semantics of DSMML named XMML designed by us based on first-order logic, based on this, the approach of consistency verification of XMML and metamodels is presented, and then design and implementation of corresponding formalization automatic mapping engine for metamodels is introduced to show the application of formalization of XMML.

II. RELATED WORKS

Within the domain-specific language community, graph-theoretic formalisms have received the most research attention [4]. The majority of work focuses on model transformations based on graph, but analysis and validation of properties of models has not received the same attention. For example, the model transformation tool VIATRA [5] supports executable Horn logic to specify transformations, but does not focus on restricting expressiveness for the purpose of analysis.

Because UML includes many diagrams including metamodeling, state machines, activities, sequence charts and so on, approaches for formalizing UML must tackle the temporal nature of its various behavioral semantics, necessitating more expressive formal methods. All these approaches must make trade-offs between expressiveness and the degree of automated analysis. For example, Z [6] or B [7] formalizations of UML could be a vehicle for studying rich syntax, but automated analysis and verification is less likely to be found.

Supported by Yunnan Provincial Department of Education Research Fund Key Project (No. 2011z025) and General Project (No. 2011y214)
Corresponding author: E-mail addresses: jtzwy123@gmail.com

doi:10.4304/jsw.7.6.1321-1328

There are much typical work on formalization of modeling language, such as Andre's formalization and verification of UML class diagram based on ADT [8], Kaneiwa's formalization of UML class diagram based on first-order logic [9], Paige's formalization of BON based on PVS [10] and Jackson.E.K's formalization of DSML based on Horn logic [11] and so on. Without considering formalization of metamodeling language and automatic translation from metamodels to the corresponding formal semantic domain, these approaches have lower level of automated analysis and verification.

III. AN INTRODUCTION TO XMML

We begin by introducing layered architecture of XMML and then an overview of abstract syntax of XMML is showed.

A. Layered Architecture of XMML

Similar to structure of UML, XMML is divided into the following four layers: metamodeling language layer used to define different DSMLs where XMML is located, DSML layer used to build concrete domain application models, domain application model layer used to make corresponding source codes of target system by code generator, and target application system layer [12]. Layered Architecture of XMML is shown in Figure 1.

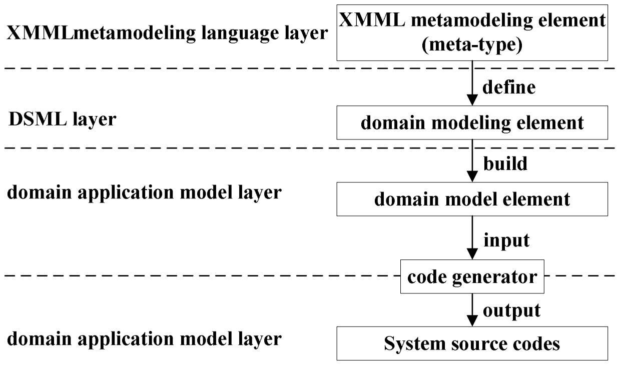


Figure 1. Layered architecture of XMML

In order to distinguish between model elements of different levels of abstraction, we require that element of XMML is called metamodeling element and element of DSML built based on metamodeling is called domain modeling element and domain object built based on domain application modeling is called domain model element. Among them, metamodeling element is also called meta-type and type of model element is name of modeling element and type of modeling element is name of meta-type.

B. Abstract Syntax of XMML

We extend and refine abstract syntax of XMML to meet the needs of formalization and consistency verification. Metamodeling element of improved XMML is divided into two types: entity type and association type, the former is used to describe modeling entities in domain metamodel and the latter concerns relationships between modeling entities.

Metamodeling element of entity type contains four types such as model type, entity type, reference entity type and relationship type; metamodeling element of association type includes the following six types: role assignment association used to establish the connection between the entity type and relationship type, model containment relationship used to build the relationship that model contain all entity type modeling elements, attachment relationship used to describe close containment relationship between entity type modeling elements, entity containment relationship used to describe loose containment relationship between entity type modeling elements, reference relationship used to build reference between reference entity and referenced entity, and refinement relationship used to establish correspondence between the entity and its refined model for multi-layer modeling and model refinement. The structural semantics of XMML will be formalized based on the above ten types of metamodeling elements.

IV. FORMALIZATION OF XMML BASED ON FIRST-ORDER LOGIC

We give a formal definition of XMML, based on this, attachment relationship, refinement relationship and typed constraints of XMML is formalized based on first-order logic to show our approach for formalizing the structural semantics of XMML.

A. A Formal Definition of XMML

XMML can be regarded as composition of the following five parts: a set of predicate symbols $\mathcal{S}_{XMML}$ denoting corresponding metamodeling elements, an extended set of predicate symbols $\mathcal{S}^{C}_{XMML}$ used to derive properties, a set of closed first-order logic formulas $\mathcal{F}_{XMML}$ denoting constraints over all metamodels built based on XMML, a set of constants O_{XMML} denoting public properties, a set of terms symbols Ω_{XMML} denoting modeling elements constituting metamodel. Among them, $\mathcal{S}^{C}_{XMML}$ and O_{XMML} may be empty, $\mathcal{F}_{XMML}$ is defined using first-order logic implication formulas based on $\mathcal{S}_{XMML}$, $\mathcal{S}^{C}_{XMML}$ and O_{XMML}. The definition concerns formal characterization of structural properties of XMML, focusing on description of constraint relationship between modeling elements. So XMML is defined as following.

Definition 1 (**XMML**). DSMML named XMML $\mathcal{L}_{XMML}$ is a 5-tuple of the form $\langle \mathcal{S}_{XMML}, \mathcal{S}^{C}_{XMML}, \Omega_{XMML}, O_{XMML}, \mathcal{F}_{XMML}, \rangle$ consisting of $\mathcal{S}_{XMML}$, $\mathcal{S}^{C}_{XMML}$, O_{XMML}, $\mathcal{F}_{XMML}$ and Ω_{XMML}.

$\mathcal{S}_{XMML}$ and $\mathcal{S}^{C}_{XMML}$ as a group of predicate symbols, O_{XMML} as a group of constant symbols, and $\mathcal{F}_{XMML}$ as a group of constraint axioms are all added to first-order logic formalized system called predicate calculus Q [13][14] to form formalized system of XMML called T_{XMML} based on predicate calculus Q. The powerset of the term algebra $\mathcal{M}_{XMML} = \mathcal{P}(T_{\mathcal{S}_{XMML}}(\Sigma_{XMML}))$ over $\mathcal{S}_{XMML}$ generated by $\Sigma_{XMML} = \Omega_{XMML} \cup O_{XMML}$ is considered as a group of interpretations of T_{XMML} to determine whether any metamodel $m_{XMML} \in \mathcal{M}_{XMML}$ is well-formed for XMML. Once $\mathcal{S}_{XMML}$, $\mathcal{S}^{C}_{XMML}$, O_{XMML} and $\mathcal{F}_{XMML}$ are derived, we finish formalization of $\mathcal{L}_{XMML}$ based on first-order logic.

B. Formalization of Meta-type of Entity Type

For each *Model*, a unary predicate $Model(x)$ is defined to denote meta-type of modeling element *x* is *Model*, i.e. $Model(x) \in \mathcal{S}_{XMML}$. *Model* can contain other two modeling elements of entity type. For each *Entity*, a unary predicate $Entity(x)$ is defined to denote meta-type of modeling element *x* is *Entity*, i.e. $Entity(x) \in \mathcal{S}_{XMML}$. *Entity* can be contained in *model* by model containment relationship or point to refined *model* by refinement relationship or establish association with other *entity* by role assignment association or form containment with other entity by attachment relationship or entity containment relationship. For each *Reference Entity*, a unary predicate $RefEntity(x)$ is defined to denote meta-type of modeling element *x* is *Reference Entity*, i.e. $RefEntity(x) \in \mathcal{S}_{XMML}$. *Reference Entity* can point to referenced entity by reference relationship. Similarly, for each *Relationship*, a unary predicate $Relationship(x)$ is defined to denote meta-type of modeling element *x* is *Relationship*, i.e. $Relationship(x) \in \mathcal{S}_{XMML}$. *Relationship* can be used to establish explicit association between modeling element of entity type combined with *role assignment association*.

C. Formalization of Attachment Relationship

For each attachment relationship (denoted *Attachment*) from modeling element of entity type *x* to *y*, a binary predicate *Attachment*(*x*,*y*) is defined to represent that element *x* is attached to element *y*, i.e. $Attachment(x,y) \in \mathcal{S}_{XMML}$. In the metamodel shown in Figure 2, modeling element of entity type *Interface* is attached to *Component*, so *Attachment*(*Interface*,*Component*) is a legal binary predicate symbol of attachment meta-type. As can be seen from Figure 3, there exist the following several constraint relationships.

1) *Type Constraint:* Attachment edge must start from and also end with modeling element of entity type. This can be expressed as an implication formula named *Attach1* in the form of $\forall x,y.Attachment(x,y) \rightarrow Entity(x) \wedge Entity(y)$.

2) *Self-attached Constraint:* Due to its close containment, the same modeling element of entity type cannot be attached to itself. For example, self-attached of *Interface* in Figure 4 is not allowed. We can express this as a predicate formula named *Attach2* in the form of $\forall x.\neg Attachment(x,x)$.

3) *Attachment Loop:* Attachment loop formed between two modeling elements of entity type is not allowed because it expresses a contradictory and meaningless modeling intent. For example, attachment loop between *Interface* and *Component* in Figure 5 is illegal. This can be expressed as an implication formula named *Attach3* in the form of $\forall x,y.Attachment(x,y) \rightarrow \neg Attachment(y,x)$.

4) *Attachment Path:* To maintain well-formedness and reduce the complexity, we require that only one layer of attachment path between two entities is legal and two or more layers of attachment path formed between entities are prohibited. For example, two layers of attachment path formed by *Interface* attached to *Component* and *Component* attached to *Subsystem* in Figure 6 is not allowed. Assume that two layers of attachment path formed by *x* attached to *y* and *y* attached to *z* is denoted as *AttaPath*(*x*,*y*,*z*), i.e. $AttaPath(x,y,z) \in \mathcal{S}^{C}_{XMML}$, *AttaPath*(*x*,*y*,*z*) can be defined by *Attachment* as an implication formula in the form of $\forall x,y,z.Attachment(x,y) \wedge Attachment(y,z) \wedge (x \neq y) \wedge (y \neq z) \wedge (x \neq z) \rightarrow AttaPath(x,y,z)$, so we can express this constraint as a predicate formula named *Attach4* in the form of $\forall x,y,z.\neg AttaPath(x,y,z)$.

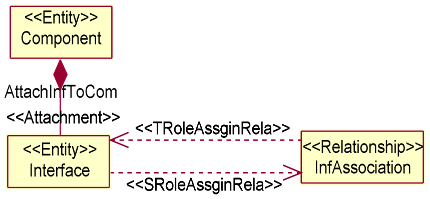


Figure 2. An example of metamodel

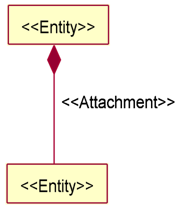


Figure 3. Attachment

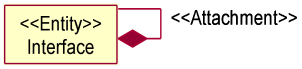


Figure 4. Self-attached

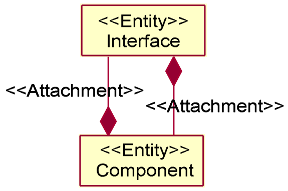


Figure 5. Attachment loop

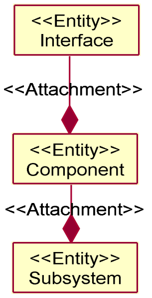


Figure 6. Attachment path

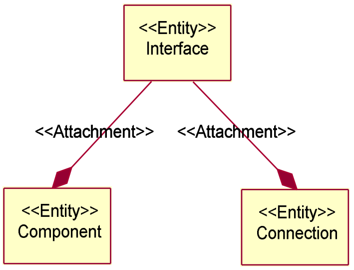


Figure 7. An example of attachment

According to the *Attach1*, both ends connected by attachment edge are all modeling elements of entity type, thus, from the perspective the semantics of first-order logic, we can prove the semantic non-implication from *Attach1 to Attach2*, *Attach3* and *Attach4* by finding a counter-example interpretation that makes *Attach1* true and makes *Attach2, Attach3 and Attach4* false.

Theorem 1 (**Semantic non-implication of attachment constraint**). Formula *Attach1* cannot semantically entail formula *Attach2, Attach3 and Attach4*, i.e. *Attach1* $|\neq$ *Attach2*, *Attach1* $|\neq$ *Attach3* and *Attach1* $|\neq$ *Attach4*.

Proof. As a semantic interpretation of formula set composed of *Attach1* ~*Attach4*, the metamodel shown in Figure 4 can be expressed as a set of predicate statements composed of *Attachment*(*Interface*,*Interface*) and *Entity*(*Interface*) that makes *Attach1* true and makes *Attach2* false due to self-attached of *Interface*, so we can derive *Attach1* $|\neq$ *Attach2*. Similarly, the metamodel shown in Figure 5 can be expressed as a set of predicate statements composed of *Attachment*(*Interface*,*Component*) and *Attachment*(*Component*,*Interface*) that makes *Attach1* true and makes *Attach3* false due to attachment loop formed between *Interface* and *Componennt*, thus, *Attach1* $|\neq$ *Attach3* can be derived. In addition, *Attachment*(*Interface*,*Component*) and *Attachment*(*Component*,*Subsystem*) corresponding to the metamodel in Figure 6 all satisfy *Attach1* but make *Attach4* false due to two layers of attachment path formed among *Interface*, *Componennt* and *Subsystem*, therefore, we can derive *Attach1* $|\neq$ *Attach4*.

Are there grammatical inference relationships among *Attach2*, *Attach3* and *Attach4*? We find that *Attach2* can be derived from *Attach3* based on natural deduction rules for quantifiers (NDRQ) which include premise introduction rule (denoted P), separation rule (denoted S) , return false rule (denoted N) and quantifier rule (denoted Q) and so on [14]. Therefore, we can derive the following theorem.

Theorem 2 (**Grammatical inference relationship of attachment constraint**). Formula *Attach2* can be derived from Formula *Attach3*, i.e.

$$\forall x, y. Attachment(x, y) \rightarrow \neg Attachment(y, x) \top \ \forall x. \neg Attachment(x, x)$$

Proof. (Derivation is omitted).

Because of *Attach3* $\vdash$ *Attach2*, after *Attach2* is removed, there are only *Attach1*, *Attach3* and *Attach4* among which there are 6 pairs of semantic non-implication relations. Similar to theorem 1, we can also derive *Attach3* $|\neq$ *Attach4*, *Attach3* $|\neq$ *Attach1*, *Attach4* $|\neq$ *Attach1* and *Attach4* $|\neq$ *Attach3*, so it is obvious that *Attach1*, *Attach3* and *Attach4* are independent on semantics. Therefore, the formula set of attachment constraints only contains *Attach1*, *Attach3* and *Attach4*.

Theorem 3 (**Semantic consistency of formula set**). The formula set comprised of *Attach1*, *Attach3* and *Attach4* is semantic consistent.

Proof. As a semantic interpretation of formula set composed of *Attach1*, *Attach3* and *Attach4*, the metamodel shown in Figure 7 can be expressed as a set of predicate statements composed of *Attachment*(*Interface*,*Component*) and *Attachment*(*Interface*,*Connection*). Because there do not exist attachment loops and two or more layers of attachment paths in the metamodel, both of them all satisfy *Attach1*, *Attach3* and *Attach4*. Therefore, the metamodel shown in Figure 7 can be considered as a semantic interpretation that satisfies the formula set, or the formula set is satisfiable. By related definitions of first-order logic, theorem is proved.

According to related theorems of first-order logic [14], the formula set is grammatical consistent, thus, it is consistent. So the formula subset of attachment constraints named *AttachmentSet* is comprised of *Attach1*, *Attach3* and *Attach4*, i.e. *AttachmentSet*= {*Attach1*, *Attach3*, *Attach4*}.

D. Formalization of Refinement Relationship

For each refinement relationship (denoted *Refinement*) from modeling element of entity type *x* to model type *y*, a binary predicate *Refinement*(*x*,*y*) is defined to represent that element *x* points to element *y* by *refinement* edge, i.e. $Refinement(x, y) \in \mathcal{S}_{XMML}$. In the metamodel shown in Figure 8, the edge *Refinement*(*Component*, *SoftwareArchitecture*) built by modeling element of entity type *Component* pointing to its refined model *SoftwareArchitecture* is a legal binary predicate symbol of *refinement* meta-type. As can be seen from Figure 9, there exist the following several constraint rules.

1) *Type Constraint:* refinement edge must start from modeling element of entity type and end with modeling element of model type. This can be expressed as an implication formula named *Refine1* in the form of $\forall x, y. Refinement(x, y) \rightarrow Entity(x) \wedge Model(y)$.

2) *Uniqueness Constraint:* the same modeling element of entity type cannot point to two or more refined models, otherwise ambiguity will be produced. For example, the metamodel in Figure 13 is illegal because the modeling element *Component* points to two different refined models *SoftwareArchitectureA* and *SoftwareArchitectureB*. We can express this as an implication formula named *Refine2* in the form of $\forall x, y, z. Refinement(x, y) \wedge Refinement(x, z) \rightarrow (y = z)$.

3) *Identity Constraint:* the refined model that the modeling element of entity type points to and the model in which it is contained are identical to build multi-layer model structure using recursive relationship. For example, in Figure 14, the refined model *SoftwareArchitectureB* of *Component* and the model *SoftwareArchitectureA* containing it are different, so multi-layer model structure cannot be built based on it. This can be expressed as an implication formula named *Refine3* in the form of $\forall x, y, z. Refinement(x, y) \wedge Containment(x, z) \rightarrow (y = z)$. In formula *Refine3*, $Containment(x, y)$ is a binary predicate denoting model containment relationship in which modeling element of entity type *x* is contained in model type *y*.

4) *Self-refinement Constraint:* the same modeling element of entity type cannot point to itself by *refinement* edge. For example, self-refinement of *Component* in Figure 10 is not allowed. We can express this as a predicate formula named *Refine4* in the form of $\forall x. \neg Refinement(x, x)$.

5) *Refinement Loop Constraint:* the refinement loop formed between two modeling elements is not allowed because it expresses a contradictory and meaningless

modeling intent. For example, refinement loop formed by *Component* and *SoftwareArchitecture* pointing to each other in Figure 11 is illegal. This can be expressed as an implication formula named *Refine5* in the form of $\forall x, y.Refinement(x, y) \wedge (x \neq y) \rightarrow \neg Refinement(y, x)$.

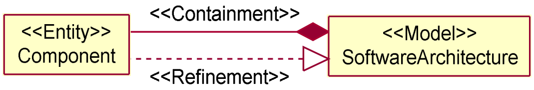


Figure 8. An example of metamodel

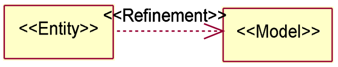


Figure 9. Refinement

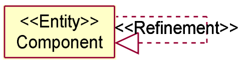


Figure 10. Self- refinement

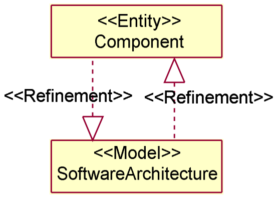


Figure 11. Refinement loop

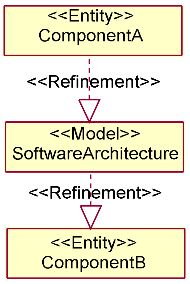


Figure 12. Refinement path of two layers

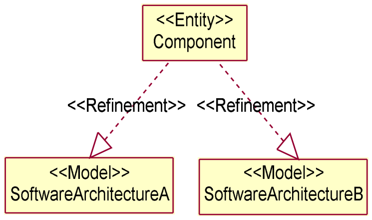


Figure 13. Refinement ambiguity

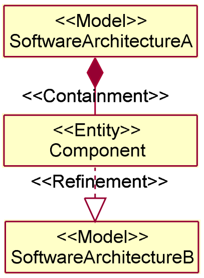


Figure 14.Refined and containing model

Figure 15. An example of violating *Refine1*

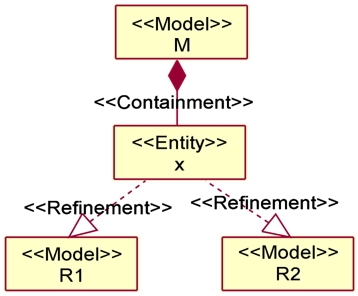


Figure 16. *Refine3* and *Con5* deriving *Refine2*

6) *Refinement Path Constraint:* To maintain well-formedness and reduce the complexity, we require that only one layer of refinement path between two entities is legal and two or more layers of refinement path are prohibited. For example, two layers of refinement path formed by *ComponentA* pointing to *SoftwareArchitecture* and *SoftwareArchitecture* pointing to *ComponentB* in Figure 12 is not allowed. Assume that two layers of refinement path formed by *x* pointing to *y* and *y* pointing to *z* is denoted as $RefinePath(x, y, z)$, i.e. $RefinePath(x, y, z) \in \mathcal{S}^{C}_{XMML}$, $RefinePath(x, y, z)$ can be defined by *refinement* as an implication formula in the form of $\forall x, y, z.Refinement(x, y) \wedge Refinement(y, z) \wedge (x \neq y) \wedge (y \neq z) \wedge (x \neq z) \rightarrow RefinePath(x, y, z)$, so we can express this constraint as a predicate formula named *Refine6* in the form of $\forall x, y, z.\neg RefinePath(x, y, z)$.

Any modeling element belongs to one and only one meta-type, on the other hand, according to the *Refine1*, both ends connected by refinement edge belong to different meta-type, therefore from the perspective the semantics of first-order logic, we can prove the semantic implication from *Refine1 to Refine4*, *Refine5* and *Refine6*.

Theorem 4 (Semantic implication of refinement constraint). Formula *Refine1* can semantically entail formula *Refine4, Refine5 and Refine6,* i.e. *Refine1* $\models$*Refine4*, *Refine1* $\models$*Refine5* and *Refine1* $\models$*Refine6*.

Proof. Any semantic interpretation that makes *Refine1* true prompts *refinement* to satisfy the relationship that both ends of *refinement* belong to different meta-type in which one end is modeling element of entity type and the other end is modeling element of model type; obviously, the relationship excludes the possibility of self-refinement of the same modeling element and also makes it impossible to form refinement loop and two or more layers of refinement path, thus, this interpretation certainly makes *Refine4, Refine5 and Refine6* true, according to related definition of semantic implication of first-order logic, theorem is proved.

Now formula set of refinement constraints contains only *Refine1*, *Refine2* and *Refine3*, are there semantic implication relationships among them? We find that *Refine2* can be derived by identity of refinement named *Refine3* and uniqueness of the models in which the same modeling element of entity type is contained named *cont5*. In Figure 16, the modeling element *x* points to two different refined models R_1 and R_2 by two different refinement edges and the model *M* that can contain *x* is unique by *cont5*, thus by *Refine3* R_1 and *M* are the same modeling element of model type, i.e. $R_1=M$, similarly, R_2 and *M* are the same modeling element of model type, i.e. $R_2=M$, so $R_1= R_2$.

Only *Refine1* and *Refine3* left in the set and among them there are two kinds of semantic implication relationship. Although syntax derivation between them cannot be directly proved, we can explain the semantic non-implication from *Refine1 to Refine3* by finding a counter-example interpretation that makes *Refine1* true and makes *Refine3* false from the perspective the semantics of first-order logic. Similarly, the semantic non-implication from *Refine3 to Refine1* can also be explained.

Theorem 5 (**Semantic non-implication of refinement constraint**). Formula *Refine1* cannot semantically entail formula *Refine3*, otherwise the same, i.e. $Refine1 \mid\neq Refine3$ and $Refine3 \mid\neq Refine1$.

Proof. As a semantic interpretation of formula set composed of *Refine1* and *Refine3*, the metamodel shown in Figure 14 can be expressed as a set of predicate statements composed of *Refinement*(*Component*,*SoftwareArchitectureB*) and *Containment*(*Component*,*SoftwareArchitectureA*) that makes *Refine1* true and makes *Refine3* false due to violation of identity constraint, so we can derive $Refine1 \mid\neq Refine3$. Similarly, the metamodel shown in Figure 15 can be expressed as a set of predicate statements composed of *Refinement*(*ComponentA*,*ComponentB*) and *Containment*(*ComponentA*,*ComponentB*) that makes *Refine3* true and makes *Refine1* false due to violation of type constraint, so $Refine3 \mid\neq Refine1$ can be derived.

Theorem 6 (**Semantic consistency of formula set**). The formula set comprised of *Refine1* and *Refine3* is semantic consistent.

Proof. As a semantic interpretation of formula set composed of *Refine1* and *Refine3*, the metamodel shown in Figure 8 can be expressed as a set of predicate statements composed of *Refinement*(*Component*,*SoftwareArchitecture*) and *Containment*(*Component*,*SoftwareArchitecture*). Because *Compoment* belongs to entity type and *SoftwareArchitecture* is an element of model type and the refined model that *Compoment* points to and the model in which it is contained are same *SoftwareArchitecture*, both of them all satisfy *Refine1* and *Refine3*. Therefore, the metamodel shown in Figure 8 can be considered as a semantic interpretation that satisfies the formula set, or the formula set is satisfiable. By related definitions of first-order logic, theorem is proved.

According to related theorems of first-order logic [12], the formula set is grammatical consistent, thus, it is consistent. So the formula subset of refinement constraints named *RefinementSet* is comprised of *Refine1* and *Refine3*, i.e. $RefinementSet = \{Refine1, Refine3\}$.

E. Formalization of Typed Constraints

XMML is one of typed metamodeling languages, so the metamodels built based XMML are well-typed. On the basis of the relevant literatures [15], we characterize typed constraints of XMML from completeness and uniqueness of classification of modeling elements of entity type.

1) Completeness of Classification

Four types of meta-types of entity type are defined in XMML to build metamodels, so their classification is also established. Such a classification must be complete in the sense that every modeling element of entity type must be an instance of one meta-type of entity type; otherwise the metamodel will become meaningless because it contains elements not belonging to any meta-type of entity type. We can express this as a predicate formula named *Type1* in the form of $\forall x.Model(x) \vee Entity(x) \vee Relationship(x) \vee RefEntity(x)$. *Type1* denotes that any modeling element of entity type must belong to one of the above four meta-types.

2) Uniqueness of Classification

The classification of modeling elements of entity type must be unique because allowing a modeling element to belong to more than one meta-type leads to ambiguous interpretation of the element, so we require that every modeling element belongs to one and only one meta-type. This can be expressed as a group of implication formulas named *Type2* in the form of

$$\begin{array}{l} \forall x.Model(x) \to \neg Entity(x) \\ \forall x.Model(x) \to \neg Relationship(x) \\ \forall x.Model(x) \to \neg RefEntity(x) \\ \forall x.Entity(x) \to \neg Relationship(x) \\ \forall x.Entity(x) \to \neg RefEntity(x) \\ \forall x.Relationship(x) \to \neg RefEntity(x) \end{array} .$$

Number of formulas *numtype2* in *Type2* is combination number produced by taking any two elements from six elements, i.e. $numtype2=4\times(4-1)\times 0.5=6$. So the formula subset of typed constraints named *TypedSet* is comprised of *Type1* and *Type2*, i.e. $TypedSet = \{Type1, Type2\}$.

TypedSet makes it explicit that a metamodel as an instance of XMML must have its modeling elements of entity type completely and uniquely classified by four types of meta-types. This reflects strict meta-modeling principle proposed in the literature [15].

F. Formalization of Other Meta-type of Association Type

By formalizing other meta-types of association type in the same way, we can establish formula subset of role assignment association constraints named *RoleAssginRelaSet*, formula subset of model containment constraints named *ContainmentSet*, formula subset of entity containment constraints named *EntiContSet* and formula subset of reference constraints named *ReferenceSet* one by one. Based on this, formula subset of exclusion constraints named *ExclusionSet* is created to represent exclusive constraints among all meta-types. Therefore, set of constraint axioms of T_{XMML} named $\mathscr{F}_{XMML}$ can be considered as union of all of the above subsets, i.e.

$$\begin{array}{l} \mathscr{F}_{XMML} = ContainmentSet \cup AttachmentSet \cup EntiContSet \cup \\ \qquad RoleAssginRelaSet \cup RefinementSet \cup \\ \qquad ReferenceSet \cup ExclusionSet \cup TypedSet. \end{array}$$

V. CONSISTENCY AND VERIFICATION OF XMML AND ITS METAMODELS

Formalized system of XMML called T_{XMML} based on predicate calculus Q is established by formalization of all meta-types of XMML. The semantic interpretation of T_{XMML} is a metamodel built based on XMML, universe of discourse of interpretation is the set of all entity modeling elements and constants contained in the metamodel. Similarly, metamodel built based on XMML can be formalized via metamodel mapping from metamodel to a set of predicate statements.

Once XMML and metamodel are formalized based on first-order logic, we can implement logical consistency verification of XMML and its metamodel based on first-order logical inference.

A. Consistency and Verification of XMML

It is not easy to find a true interpretation for constraint axiom set $\mathscr{F}_{XMML}$ of T_{XMML} to prove semantic consistency of T_{XMML} , on the other hand, It is very difficult to derive grammatical consistency of $\mathscr{F}_{XMML}$ by hand-proving due to too many formulas contained in $\mathscr{F}_{XMML}$, so we can only prove logical consistency of T_{XMML} based on automatic theorem prover. Reference to the literature [15], we give the following definition.

Definition 2 (**logical consistency of XMML**). XMML is logically consistent iff the constraint axiom set $\mathscr{F}_{XMML}$ of T_{XMML} is proved to be logically consistent in the automatic theorem prover; XMML is logically inconsistent iff the constraint axiom set $\mathscr{F}_{XMML}$ of T_{XMML} is proved to be contradictory in the automatic theorem prover, denoted $\mathscr{F}_{XMML} \vdash False$.

B. Consistency and Verification of Metamodel

If T_{XMML} is proved to logically consistent, then XMML must have an interpretation that can be satisfied, thus it is meaningful to discuss properties of metamodels built based on XMML. From the point of view of formalization, a legal metamodel is an interpretation that satisfies all constraint formulas of $\mathscr{F}_{XMML}$, so the relationship that metamodel satisfies XMML is equivalent to the relationship that the interpretation of T_{XMML} satisfies T_{XMML}. By equivalence of satisfaction relationship and logical consistency, we can obtain determination method of consistency of metamodel built based on XMML.

Inference 1 (**logical consistency of metamodel**). If union of constraint axiom set $\mathscr{F}_{XMML}$ of T_{XMML} and set of first-order predicate statements $T_L(M)$ generated via metamodel M is logically consistent, then the metamodel M is consistent; instead, if union of constraint axiom set $\mathscr{F}_{XMML}$ of T_{XMML} and set of first-order predicate statements $T_L(M)$ generated via metamodel M is logically inconsistent, denoted $\mathscr{F}_{XMML} \cup T_L(M) \vdash False$, then the metamodel M is inconsistent.

VI. DESIGN AND IMPLEMENTATION OF *MAPM*

Formalization automatic mapping engine for metamodel called *MapM* (*Mapping of Metamodels*) is designed and implemented to finish automatic translation from metamodel based on *XMML* concrete syntax scheme to the corresponding set of first-order predicate statements $T_L(M)$ in SPASS format [16], thus we can realize automatic process of analysis and verification of consistency of metamodel built based on XMML. Logical architecture of *MapM* is shown in Figure 17.

Based on .net 2.0 platform, by using C#.net as development language, we implement the corresponding prototype system for *MapM* and integrate them in the modeling environment named *Archware* [12] of XMML, thus it becomes possible for *Archware* to verify metamodels built based on XMML. Running interface of *MapM* is shown in Figure 18, its left window shows XML format document of metamodel produced by *Archware* and the corresponding first-order logic system in SPASS format generated by translation of *MapM* is showed in right window.

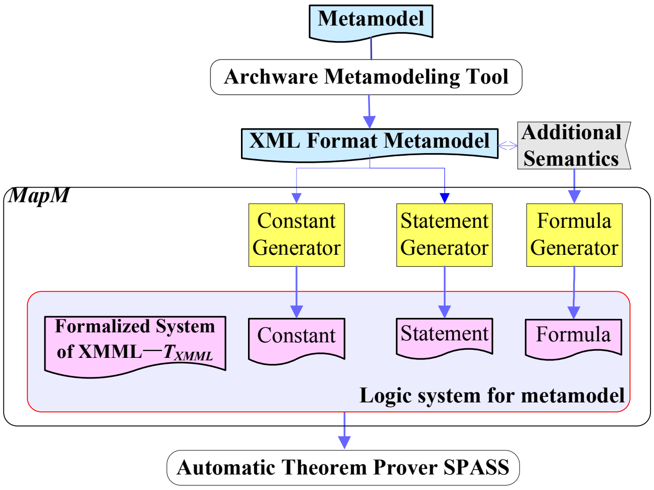
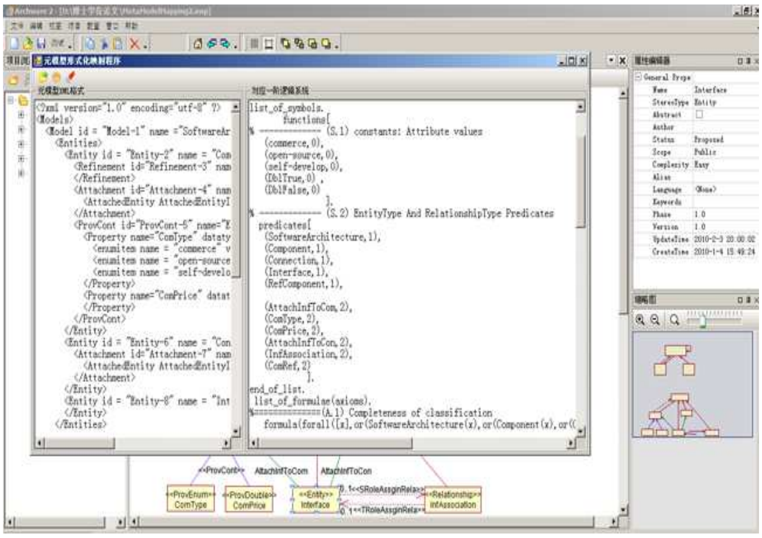


Figure 17. Logical architecture of *MapM*



```
<?xml version="1.0" encoding="utf-8" ?>
<Models>
  <Model id = "Model-1" name ="SoftwareAr
    <Entities>
      <Entity id = "Entity-2" name = "Com
        <Refinement id="Refinement-3" nam
        </Refinement>
        <Attachment id="Attachment-4" nam
          <AttachedEntity AttachedEntityI
        </Attachment>
        <ProvCont id="ProvCont-5" name="E
          <Property name="ComType" dataty
            <enumitem name = "commerce" v
            <enumitem name = "open-source
            <enumitem name = "self-develo
          </Property>
          <Property name="ComPrice" datat
          </Property>
        </ProvCont>
      </Entity>
      <Entity id = "Entity-6" name = "Con
        <Attachment Id="Attachment-7" nam
          <AttachedEntity AttachedEntityI
        </Attachment>
      </Entity>
      <Entity id = "Entity-8" name = "Int
      </Entity>
    </Entities>
```

```
list_of_symbols.
      functions[
% ------------ (S.1) constants: Attribute values
   (commerce,0),
   (open-source,0),
   (self-develop,0),
   (DblTrue,0) ,
   (DblFalse,0)
              ].
% ------------ (S.2) EntityType And RelationshipType Predicates
  predicates[
   (SoftwareArchitecture,1),
   (Component,1),
   (Connection,1),
   (Interface,1),
   (RefComponent,1),

   (AttachInfToCom,2),
   (ComType,2),
   (ComPrice,2),
   (AttachInfToCon,2),
   (InfAssociation,2),
   (ComRef,2)
            ].
end_of_list.
 list_of_formulae(axioms).
%=============(A.1) Completeness of classification
   formula(forall([x],or(SoftwareArchitecture(x),or(Component(x),or((
```

General Props	
Name	Interface
StereoType	Entity
Abstract	☐
Author	
Status	Proposed
Scope	Public
Complexity	Easy
Alias	
Language	<None>
Keywords	
Phase	1.0
Version	1.0
UpdateTime	2010-2-3 20:00:02
CreateTime	2010-1-4 15:49:24

Figure 18. Running interface of *MapM*

VII. CONCLUSIONS

The paper's work derives from Yunnan Province Department of Education Research Fund Key Project (No.2011z025). DSMML defined in the informal way cannot precisely describe its structural semantics, which makes it difficult to systematically verify its properties such as consistency. In response, the paper proposes a formal representation of the structural semantics of DSMML named XMML designed by us based on first-order logic. And then we illustrate our approach by formalization of attachment relationship and refinement relationship and typed constraints of XMML based on first-order logic. Based on this, the approach of consistency verification of XMML itself and metamodels is presented. Finally, we design and implement the corresponding formalization automatic mapping engine for metamodel to show the application of formalization of XMML.

ACKNOWLEDGMENT

The author would like to thank Prof. Hua Zhou, Dr. Xinping Sun and Dr. Yong Yu for valuable discussions. This work was supported by Yunnan Provincial Department of Education Research Fund Key Project (No. 2011z025) and General Project (No. 2011y214).

REFERENCES

[1] Miller J, Mukerji J, MDA guide version *1.0.1*. http://www.omg.org/ docs/omg/03- 06-01.pdf, 2003.
[2] dsmforum, Enterprise apps in smartphones, http://www.dsmforum.org/phone.html..
[3] Jackson.E.K, Sztipanovits.J, “Formalizing the Structural Semantics of Domain-Specific Modeling Languages”, Journal of Software and Systems Modeling, 2008.
[4] B´E ZIVIN, J., AND GERB´E, O, “Towards a precise definition of the omg/mda framework”, in Proceedings of the 16th Conference on Automated Software Engineering (ASE 01) (2001), pp. 273–280.
[5] CSERT´A N, G., HUSZERL, G., MAJZIK, I., PAP, Z., PATARICZA,A., AND VARR´O, D. “Viatra - visual automated transformations for formal verification and validation of uml models”, in ASE (2002), pp. 267–270.
[6] EVANS, A., FRANCE, R. B., AND GRANT, E. S, “Towards formal reasoning with uml models”, in Proceedings of the Eighth OOPSLA Workshop on Behavioral Semantics.
[7] MARCANO, R., AND LEVY, N, “Using b formal specifications for analysis and verification of uml/ocl models”, in Workshop on consistency problems in UML-based software development.5th International Conference on the Unified Modeling Language (2002), pp. 91–105.
[8] W.Andreopoulos, “Defining Formal Semantics for the Unified Modeling Language”, in Technique Report of University of Toronto[C], 2000, Toronto.
[9] K. Kaneiwa and K. Satoh, “Consistency Checking Algorithms for Restricted UML Class Diagrams”, in 4th International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2006)[C], LNCS 3861, 2006. p.19 - 239.
[10] R. F. Paige, B. P. J, “Metamodel-Based Model Conformance and Multiview Consistency Checking”, ACM Transactions on Software Engineering and Methodology, 2007. 16(3): p. 1-49.
[11] Jackson.E.K, Sztipanovits.J, “Towards a formal foundation for domain specific modeling languages”, Proceedings of the Sixth ACM International Conference on Embedded Software (EMSOFT'06) (October 2006) 53-62.
[12] Sun XP, A Research of Visual Domain-Specific Meta-Modeling Language and Its Instantiation, Kunming: Yunnan University.2008.
[13] Gu TL, Formal methods of software development, Higher Education Press, Beijing, 2005.
[14] Cheng MZ, Yu JW, Logic foundation—first-order logic and first-order theory, Chinese People University Press, Beijing, 2003.
[15] H. Zhu, L. Shan, I. Bayley, and R. Amphlett, “A Formal Descriptive Semantics of UML and Its Applications”, in UML 2 Semantics and Applications, K. Lano (Eds). 2008, John Wiley & Sons, Inc.
[16] Christoph Weidenbach, SPASS: Tutorial, 2000.

Tao Jiang was born in Kunming, China, in 1973. He received his B.Sc. degree in Computer Software from Nanjing University, China in 1995 and received his M.Sc. degree in Computer Software and Theory from Yunnan University, China in 2003 and received his Ph.D. degree in Information Systems Analysis and Integration from Yunnan University, China in 2010. The major fields of his studies involve multiple branches of Software Engineering.

During 1996-2005, he worked as a Software Engineer in the Department of Information Technology, China Construction Bank. Currently, he is an Associate Professor in School of Mathematics and Computer Science, Yunnan University of Nationalities.

His research areas cover Domain-Specific Visual Modeling, Modeling Formalization, Model Verification, Formal Method of Software Development and Web Application and so on. He has more than 20 published scientific papers in international conferences and journals.

Xin Wang was born in Kunming, China, in 1963. He received his M.Sc. degree in Software Enigeering from Yunnan University, China in 2006. The major fields of his studies involve Software Engineering and Data Mining and so on. Currently, he is a Professor in School of Mathematics and Computer Science, Yunnan University of Nationalities. His research areas cover Model Checking, Formal Method of Software Development, Database Application and Data Ming and so on. He has more than 10 published scientific papers in international conferences and journals.