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: wxkmyx@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 DSMML 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.
There are much typical work on formalization of modeling language, such as André’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.

![Layered architecture of XMML](image)

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. So XMML is defined as following.

**Definition 1**

\[
\text{XMML} = (\text{S}_{\text{XMML}}, \text{C}_{\text{XMML}}, \text{F}_{\text{XMML}})
\]

\[
\text{XMML} = \{ \text{S}_{\text{XMML}}, \text{C}_{\text{XMML}}, \text{F}_{\text{XMML}} \}
\]

XMML can be regarded as composition of the following five parts: a set of predicate symbols \( \text{S}_{\text{XMML}} \), denoting corresponding metamodeling elements, an extended set of predicate symbols \( \text{S}^{c}_{\text{XMML}} \) used to derive properties, a set of closed first-order logic formulas \( \text{F}_{\text{XMML}} \) denoting constraints over all metamodels built based on XMML, a set of constants \( \Omega_{\text{XMML}} \) denoting public properties, a set of terms symbols \( \text{O}_{\text{XMML}} \) denoting modeling elements constituting metamodel.

**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 \( \text{S}_{\text{XMML}} \), denoting corresponding metamodeling elements, an extended set of predicate symbols \( \text{S}^{c}_{\text{XMML}} \) used to derive properties, a set of closed first-order logic formulas \( \text{F}_{\text{XMML}} \) denoting constraints over all metamodels built based on XMML, a set of constants \( \Omega_{\text{XMML}} \) denoting public properties, a set of terms symbols \( \text{O}_{\text{XMML}} \) denoting modeling elements constituting metamodel. Among them, \( \text{S}^{c}_{\text{XMML}} \) and \( \Omega_{\text{XMML}} \) may be empty, \( \text{F}_{\text{XMML}} \) is defined using first-order logic implication formulas based on \( \text{S}_{\text{XMML}}, \text{S}^{c}_{\text{XMML}} \) and \( \Omega_{\text{XMML}} \). 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)**

DSML named XMML \( \mathcal{L}_{\text{XMML}} \) is a 5-tuple of the form \( \langle \text{S}_{\text{XMML}}, \text{S}^{c}_{\text{XMML}}, \Omega_{\text{XMML}}, \mathcal{F}_{\text{XMML}}, \mathcal{O}_{\text{XMML}} \rangle \) consisting of \( \text{S}_{\text{XMML}}, \text{S}^{c}_{\text{XMML}}, \Omega_{\text{XMML}}, \mathcal{F}_{\text{XMML}}, \text{and} \mathcal{O}_{\text{XMML}} \).

\( \text{S}_{\text{XMML}}, \text{S}^{c}_{\text{XMML}}, \text{and} \Omega_{\text{XMML}} \) as a group of predicate symbols, \( \mathcal{F}_{\text{XMML}} \) as a group of constant symbols, and \( \mathcal{S}_{\text{XMML}} \) 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 \( T_{\text{XMML}} \) based on predicate calculus Q. The powerset of the term algebra \( \mathcal{M}_{\text{XMML}} = \mathcal{P} (\mathcal{T}_{\text{XMML}} (\Omega_{\text{XMML}})) \) over \( \text{S}_{\text{XMML}} \) generated by \( \Sigma_{\text{XMML}} = \Omega_{\text{XMML}} \cup \mathcal{O}_{\text{XMML}} \) is considered as a
group of interpretations of $T_{XMML}$ to determine whether any metamodel $\alpha_{XMML} \in \mathcal{M}_{XMML}$ is well-formed for XMML. Once $S_{XMML}, S^\prime_{XMML}, O_{XMML}$ and $F_{XMML}$ 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. $Model(x) \in S^\prime_{XMML}$. 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. $Entity(x) \in S^\prime_{XMML}$. 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. $RefEntity(x) \in S_{XMML}$. 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. $Relationship(x) \in S_{XMML}$. 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. $Attachment(x, y) \in S^\prime_{XMML}$. 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 $\forall x, y. Attachment(x, y) \rightarrow Entity(x) \land Entity(y)$.

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 $Attachment(x, y, z)$, i.e. $Attachment(x, y, z) \in S^\prime_{XMML}$. $Attachment(x, y, z)$ can be defined by $Attachment$ as an implication formula in the form of $\forall x, y, z. Attachment(x, y) \land Attachment(y, z) \land (x \neq y) \land (y \neq z) \land (x \neq z) \rightarrow Attachment(x, y, z)$, so we can express this constraint as a predicate formula named $Attach4$ in the form of $\forall x, y, z. \neg Attachment(x, y, z)$.
counter-example interpretation that makes \textit{Attach1} true and makes \textit{Attach2, Attach3} and \textit{Attach4} false.

\textbf{Theorem 1 (Semantic non-implication of attachment constraint)}. Formula \textit{Attach1} cannot semantically entail formula \textit{Attach2, Attach3} and \textit{Attach4}, i.e. \textit{Attach1} \not\models \textit{Attach2}, \textit{Attach1} \not\models \textit{Attach3} and \textit{Attach1} \not\models \textit{Attach4}.

Proof. As a semantic interpretation of formula set composed of \textit{Attach1 \autocite{Attach4}}, the metamodel shown in Figure 4 can be expressed as a set of predicate statements composed of \textit{Attachment(Interface,Interface)} and \textit{Entity(Interface)} that makes \textit{Attach1} true and makes \textit{Attach2} false due to self-attached of \textit{Interface}, so we can derive \textit{Attach1} \not\models \textit{Attach2}. Similarly, the metamodel shown in Figure 5 can be expressed as a set of predicate statements composed of \textit{Attachment(Interface,Component)} and \textit{Attachment(Component,Interface)} that makes \textit{Attach1} true and makes \textit{Attach3} false due to attachment loop formed between \textit{Interface} and \textit{Component}, thus, \textit{Attach1} \not\models \textit{Attach3} can be derived. In addition, \textit{Attachment(Interface,Component)} and \textit{Attachment(Component,Subsystem)} corresponding to the metamodel in Figure 6 all satisfy \textit{Attach1} but make \textit{Attach4} false due to two layers of attachment path formed among \textit{Interface, Component} and \textit{Subsystem}, therefore, we can derive \textit{Attach1} \not\models \textit{Attach4}.

Are there grammatical inference relationships among \textit{Attach2, Attach3} and \textit{Attach4}? We find that \textit{Attach2} can be derived from \textit{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.

\textbf{Theorem 2 (Grammatical inference relationship of attachment constraint)}. Formula \textit{Attach2} can be derived from Formula \textit{Attach3}, i.e.

\[ \forall x, y. \text{Attachment}(x,y) \rightarrow \neg \text{Attachment}(x,x) \]

Proof. (Derivation is omitted).

Because of \textit{Attach3} \models \textit{Attach2}, after \textit{Attach2} is removed, there are only \textit{Attach1, Attach3} and \textit{Attach4} among which there are 6 pairs of semantic non-implication relations. Similar to theorem 1, we can also derive \textit{Attach3} \not\models \textit{Attach4, Attach3} \not\models \textit{Attach1, Attach4} \not\models \textit{Attach1} and \textit{Attach4} \not\models \textit{Attach3, so it is obvious that Attach1, Attach3 and Attach4 are independent on semantics. Therefore, the formula set of attachment constraints only contains \textit{Attach1, Attach3} and \textit{Attach4}.

\textbf{Theorem 3 (Semantic consistency of formula set)}. The formula set comprised of \textit{Attach1, Attach3} and \textit{Attach4} is semantic consistent.

Proof. As a semantic interpretation of formula set composed of \textit{Attach1, Attach3} and \textit{Attach4}, the metamodel shown in Figure 7 can be expressed as a set of predicate statements composed of \textit{Attachment(Interface,Component)} and \textit{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 \textit{Attach1, Attach3} and \textit{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 \textit{AttachmentSet} is comprised of \textit{Attach1, Attach3 and Attach4}, i.e. \textit{AttachmentSet} = \{\textit{Attach1, Attach3, Attach4}\}.

\textbf{D. Formalization of Refinement Relationship}

For each refinement relationship (denoted \textit{Refinement}) from modeling element of entity type \textit{x} to model type \textit{y}, a binary predicate \textit{Refinement}(x,y) is defined to represent that element \textit{x} points to element \textit{y} by \textit{refinement} edge, i.e.

\[ \textit{Refinement}(x,y) \in \mathcal{S}_{\text{meta}} \]

In the metamodel shown in Figure 8, the edge \textit{Refinement(Component, SoftwareArchitecture)} built by modeling element of entity type \textit{Component} pointing to its refined model \textit{SoftwareArchitecture} is a legal binary predicate symbol of \textit{refinement} meta-type. As can be seen from Figure 9, there exist the following several constraint rules.

1) \textbf{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 \textit{Refine1} in the form of \[ \forall x, y. \text{Refinement}(x,y) \rightarrow \text{Entity}(x) \land \text{Model}(y) \]

2) \textbf{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 \textit{Component} points to two different refined models \textit{SoftwareArchitectureA} and \textit{SoftwareArchitectureB}. We can express this as an implication formula named \textit{Refine2} in the form of \[ \forall x, y, z. \text{Refinement}(x,y) \land \text{Refinement}(x,z) \rightarrow (y = z) \]

3) \textbf{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 \textit{SoftwareArchitectureB} of \textit{Component} and the model \textit{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 \textit{Refine3} in the form of \[ \forall x, y, z. \text{Refinement}(x,y) \land \text{Containment}(x,z) \rightarrow (y = z) \].

In formula \textit{Refine3}, \textit{Containment}(x,y) is a binary predicate denoting model containment relationship in which modeling element of entity type \textit{x} is contained in model type \textit{y}.

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

5) \textbf{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 \( \text{Refine}^5 \) in the form of \( \forall x, y. \text{Refinement}(x, y) \land (x \neq y) \rightarrow \neg \text{Refinement}(y, x) \).

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 \( \text{RefinePath}(x, y, z) \), i.e. \( \text{RefinePath}(x, y, z) = \exists_{\text{Refine}} \text{RefinePath}(x, y, z) \). \( \forall x, y, z. \text{Refinement}(x, y) \land \text{Refinement}(y, z) \land (x \neq y) \land (y \neq z) \land (x \neq z) \rightarrow \text{RefinePath}(x, y, z) \), so we can express this constraint as a predicate formula named \( \text{Refine6} \) in the form of \( \forall x, y, z. \neg \text{RefinePath}(x, y, z) \).

Any modeling element belongs to one and only one meta-type, on the other hand, according to the \( \text{Refine1} \), 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 \( \text{Refine1} \) to \( \text{Refine4} \), \( \text{Refine5} \) and \( \text{Refine6} \).

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

Proof. Any semantic interpretation that makes \( \text{Refine1} \) true prompts \( \text{Refine} \) to satisfy the relationship that both ends of \( \text{Refine} \) 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 \( \text{Refine4} \), \( \text{Refine5} \) and \( \text{Refine6} \) true, according to related definition of semantic implication of first-order logic, theorem is proved.

Now formula set of refinement constraints contains only \( \text{Refine1} \), \( \text{Refine2} \) and \( \text{Refine3} \), are there semantic implication relationships among them? We find that \( \text{Refine2} \) can be derived by identity of refinement named \( \text{Refine3} \) and uniqueness of the models in which the same modeling element of entity type is contained named \( \text{cont5} \). 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 \( \text{cont5} \), thus by \( \text{Refine3} \) \( R_1 \) 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 \( \text{Refine1} \) and \( \text{Refine3} \) 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 \( \text{Refine1} \) to \( \text{Refine3} \) by finding a counter-example interpretation that makes \( \text{Refine1} \) true and makes \( \text{Refine3} \) false from the perspective the
semantics of first-order logic. Similarly, the semantic non-implication from \( \text{Refine}3 \) to \( \text{Refine}1 \) can also be explained.

**Theorem 5 (Semantic non-implication of refinement constraint).** Formula \( \text{Refine}1 \) cannot semantically entail formula \( \text{Refine}3 \), otherwise the same, i.e. \( \text{Refine}1 \not\in \text{Refine}3 \) and \( \text{Refine}3 \not\in \text{Refine}1 \).

**Proof.** As a semantic interpretation of formula set composed of \( \text{Refine}1 \) and \( \text{Refine}3 \), the metamodel shown in Figure 14 can be expressed as a set of predicate statements composed of \( \text{Refinement}((\text{Component}, \text{SoftwareArchitecture}) \cup \text{Component}, \text{SoftwareArchitectureA}) \) that makes \( \text{Refine}1 \) true and makes \( \text{Refine}3 \) false due to violation of identity constraint, so we can derive \( \text{Refine}1 \not\in \text{Refine}3 \). Similarly, the metamodel shown in Figure 15 can be expressed as a set of predicate statements composed of \( \text{Refinement}((\text{ComponentA}, \text{ComponentB}) \cup \text{ComponentA}, \text{ComponentB}) \) that makes \( \text{Refine}3 \) true and makes \( \text{Refine}1 \) false due to violation of type constraint, so \( \text{Refine}3 \not\in \text{Refine}1 \) can be derived.

**Theorem 6 (Semantic consistency of formula set).** The formula set comprised of \( \text{Refine}1 \) and \( \text{Refine}3 \) is semantic consistent.

**Proof.** As a semantic interpretation of formula set composed of \( \text{Refine}1 \) and \( \text{Refine}3 \), the metamodel shown in Figure 8 can be expressed as a set of predicate statements composed of \( \text{Refinement}((\text{Component}, \text{SoftwareArchitecture}) \cup \text{Component}, \text{SoftwareArchitecture}) \) and \( \text{Refinement}((\text{ComponentA}, \text{ComponentB}) \cup \text{ComponentA}, \text{ComponentB}) \) that makes \( \text{Refine}3 \) true and makes \( \text{Refine}1 \) false due to violation of type constraint, so \( \text{Refine}3 \not\in \text{Refine}1 \) can be derived.

**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 \( \text{RoleAssignRelaSet} \), formula subset of model containment constraints named \( \text{ContainmentSet} \), formula subset of entity containment constraints named \( \text{EntiContSet} \) and formula subset of reference constraints named \( \text{ReferenceSet} \) one by one. Based on this, formula subset of exclusion constraints named \( \text{ExclusionSet} \) is created to represent exclusive constraints among all meta-types. Therefore, set of constraint axioms of \( \text{XMML} \) named \( \mathcal{I}_{\text{XMML}} \) can be considered as union of all of the above subsets, i.e.,

\[
\mathcal{I}_{\text{XMML}} = \text{ContainmentSet} \cup \text{AttachmentSet} \cup \text{EntiContSet} \cup \text{RoleAssignRelaSet} \cup \text{ReferenceSet} \cup \text{ExclusionSet} \cup \text{TypedSet}.
\]
XMML 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 $\mathcal{F}_{XMML}$ of $T_{XMML}$ to prove semantic consistency of $T_{XMML}$, on the other hand, it is very difficult to derive grammatical consistency of $\mathcal{F}_{XMML}$ by hand-proving due to too many formulas contained in $\mathcal{F}_{XMML}$, 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 $\mathcal{F}_{XMML}$ of $T_{XMML}$ is proved to be logically consistent in the automatic theorem prover; XMML is logically inconsistent iff the constraint axiom set $\mathcal{F}_{XMML}$ of $T_{XMML}$ is proved to be contradictory in the automatic theorem prover, denoted $\mathcal{F}_{XMML} \models \text{False}$.

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 $\mathcal{F}_{XMML}$, 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 $\mathcal{F}_{XMML}$ of $T_{XMML}$ and set of first-order predicate statements $T_{I}(M)$ generated via metamodel $M$ is logically consistent, then the metamodel $M$ is consistent; instead, if union of constraint axiom set $\mathcal{F}_{XMML}$ of $T_{XMML}$ and set of first-order predicate statements $T_{I}(M)$ generated via metamodel $M$ is logically inconsistent, denoted $\mathcal{F}_{XMML} \cup T_{I}(M) \models \text{False}$, 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 $T_{I}(M)$ 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.
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. Xiping Sun and Dr. Yong Yu for valuable discussions. This work was supported by Yunnan Provincial Department of Education Research Fund Key Project (No. 2011y2025) and General Project (No. 2011y214).

REFERENCES