An Approach for Automatically Verifying Metamodels Consistency

Tao Jiang*, Yumei She, Xin Wang

School of Mathematics and Computer Science, Yunnan Minzu University, Kunming, Yunnan, 650031, China

***Abstract* — Most Domain-Specific Metamodeling Languages (DSMML) use informal method to define their semantics, so it is difficult to precisely and automatically analyze characteristics of metamodels built based on DSMML. In response, based on formalization of DSMML called XMML developed by us, the paper proposes an approach for automatically verifying metamodels consistency by an automatic mapping mechanism to automatically translate metamodels to the corresponding first-order logic system. Firstly, we briefly present our approach for formalizing XMML and its metamodels, and then, we establish an automatic mapping mechanism for formalizing metamodels, finally, we develop an automatic mapping tool for formalizing metamodels and perform relevant experiments to validate our method.**

Keywords - metamodel; consistency; structure mapping; semantic mapping

I. INTRODUCTION

DSMML is a metamodeling language used for Domain-Specific modeling [1] and also a metalanguage used for creating Domain-Specific Modeling Languages (DSMLs). As an instance built based on DSMML, DSML characterizes its concrete syntax and structural semantics using a metamodel that is the model of the model. From a semantic point of view, the metamodel represents structural semantics of the corresponding DSML.

DSMMLs have many problems that are not well addressed, such as accurate mathematical foundation for formalizing its semantics, tool-independent formal descriptions, analysis techniques based on syntax and semantics of metamodels, automatic mapping rules and its implementation and so on. There are several examples that can illustrate this. As a formal metamodelling language, KM3 [2] cannot perform analysis of specific attributes because no expressive constraints are added to the language itself. The Generic Modeling Environment (GME) developed by Vanderbilt university [3] provide expressive DSMLs, but all its precise structural semantics completely depend on implementation of complex modeling tools. As a modeling and metamodeling standards, UML superstructure [4] and Meta-Object Facility (MOF) [5] cannot offer completely strict formal definitions for the DSML process.

In one of my papers published in the Journal of Software, we formalizes DSMML called XMML based on first-order logic, mainly discussing formalization of three relationships of XMML such as refinement and attachment and typed constraints [6], and another one of my papers published in Electrical Review proposes an approach for verifying consistency of XMML and its metamodels by first-order logic reasoning based on formalization of XMML, focusing on definition of metamodels consistency and deduction of consistency verification method [7].

Based on research results of our previously published papers, this paper proposes an automatic mapping mechanism for automatically translating metamodels to the corresponding first-order logic system based on formalization of XMML to implement automated reasoning of metamodels consistency. We define two mappings: structure mapping from one metamodel structure to a group of first-order predicate statements, and semantic mapping used to establish a group of necessary constraint formulas based on predicate statements generated via structure mapping. And then, taking software architecture metamodel MSS an example, after finishing automatic mapping of MSS, we verify its consistency using two methods of manual reasoning and automatic proving. To show the application of our formal method, we design the corresponding formalization automatic mapping tool for metamodels called MapMMD, and perform many experiments on automatic mapping of different metamodels based on MapMMD and analyze the experimental results.

II. ELATED WORKS

In UML domain, there are much typical works for formalizing and analyzing UML semantics. Lijun Shan and Hong Zhu define the semantics of metamodels in UML class diagram based on first-order logic formulas and implement a UML model translation tool to automatically reason models [8]. Their method and ours differ in the following two aspects. First, the main problem we discuss is formalization of structural semantics of metamodelling language belonging to meta-metamodel, and Shan's method focuses on modelling language belonging to metamodel. The former is more abstract and general than the latter, so their different levels of abstraction will lead to different processing method. Second, as a Domain-Specific Metamodeling Language developed by ourselves, XMML is quite different from Unified Modeling Language in its syntax definition and constitution, the core elements of XMML are entity and the relationship between entities that includes containment, reference and refinement and so on [6], but the core element of UML is class and the relationship between classes that mainly involves generalization and association, so we need to process different syntax and semantics with different formalization and mapping mechanism.

Hao Fei's provides a method of UML models formalization and verification based on description logic [9]. It only represents syntax of basic elements of UML models, without considering semantics of UML metamodel, so his verification is not sufficient and of little significance. Snook design a profile UML-B to define the semantics of specialized UML entities by mapping UML into B [10]. Moller uses combination of the process algebra CSP and the specification language Object-Z as his formal method [11]. Although these two methods all define the semantics of UML via translating models into a specific formal semantic, they only involve a certain subset of UML semantics, furthermore, automatic mapping mechanism to implement automated reasoning have not been established.

In DSM domain, many DSMLs are defined in informal way, so they cannot be systematically and comprehensively verified. For example, GME uses expanded Object Constraint Language to check models [3], and MetaEdit+ [12] of MetaCase company uses fixed declarative rules to validate model characteristics. In DSM community, the majority of formalisms method focuses on graph-theoretic model transformation. These methods all establish their own model transformation language to map the textual models to the formal domain system. For example, Matthias Tichy defines model transformations in Henshin [13], and Bernhard Rumpe uses hierarchical automata as his model transformation language [14]. Without restricting expressiveness of transformations, these methods have lower degree of automated analysis.

There are also other typical works on formalization of DSMLs. Jackson and Sztipanovits use Horn logic as formal specifications of DSML [15], and develop a theorem prover called FORMULA to reason about properties [16]. Different from their method of model transformation, we use automatic mapping to implement translation from original syntax of modeling language to formal semantics domain, in addition, we use more expressive first-order logic as our basis of formalization rather than Horn logic as a decidable subset of first-order logic.

III. FORMALIZING XMML AND ITS METAMODELS

We design a Domain-Specific metamodeling language based on extensible markup language called XMML and separate metamodeling elements of XMML into two types of entity and association. Metamodeling element of XMML is also called metatype. Entity metatype includes relationship, reference entity, entity and model. Association metatype includes seven types of model containment, entity containment, source role assignment association, target role assignment association, reference, attachment and refinement. These two metatypes of source role assignment association and target role assignment association are combined together to build the association between different entity metatypes. Model containment metatype means that all entity metatypes are included in one model. Entity containment and attachment respectively describe loose and close inclusion relationship between entity metatypes. Reference metatype can establish association reference entity pointing to referenced entity. Refinement metatype indicates corresponding relation between the entity metatype and its refined model. Details of syntax definition of XMML can be seen in [17].

After calculating union of eleven derived logical constraint formulas corresponding to the above eleven metatypes, structural semantics of XMML is formalized. Based on this, we establish formalized system of XMML called T_X. Details of formalizing XMML can be seen in [6].

After we confirm that XMML is logical consistent by automatic reasoning [6], there exist metamodels that can satisfy XMML, so it makes sense to discuss characteristics of metamodels as instances of XMML. According to theory and method of first-order logic [18], we can conclude that a metamodel is a well-formed instance of XMML if and only if the metamodel as an interpretation of formalized system of XMML called T_X satisfies all constraint formulas of T_X. Otherwise, we confirm that it is illegal. We establish a definition of interpretation of T_X and the relationship that the interpretation of T_X satisfies T_X, for more details, please see Definition 4 and Definition 5 in [7].

After formalizing XMML and its metamodels, we build the definition of metamodel consistency and then derive corollary of metamodels consistency verification by equivalence between relations that interpretation T_I of T_X satisfies T_X and logical consistency of union of T_X and predicate statements set $T_L(M)$ corresponding to T_I. Details can be seen in Definition 6, Theorem 1 and Inference 1 in [7].

IV. AN AUTOMATIC MAPPING MECHANISM FOR FORMALIZING METAMODELS

Uniqueness of XMML makes it possible for us to create formalized system T_X of XMML in artificial way, but there are many metamodels built based on XMML and they are different from each other, it is meaningless to make a formalization for every metamodel in artificial way, and this can result in full manual processing for metamodels consistency verification, so it is necessary to establish an automatic mapping mechanism from one metamodel to a corresponding first-order predicate statements set $T_L(M)$. With reference to relevant literature [8], after improving and expanding its ideas and methods for mapping UML models, we establish our automatic mapping mechanism for formalizing metamodels.

Metamodel mapping is divided into structure mapping and semantic mapping, accordingly, mapping rules are also composed of structure mapping rules and semantic mapping rules, the former is a formalization mapping from structure of one metamodel based on syntax to a first-order predicate statements set, to the latter, a first-order logic formulas set is added via the metamodel to represent some semantic constraints on the models based on the former. Structure mapping rules and semantic mapping rules will be established and illustrated by regarding the metamodel M_{SA} in Figure 1 as an example.

The metamodel M_{SA} consists of three modeling elements such as *Component*, *Interface* and *InfAssociation* that means association between *interfaces*, and it also establishes constraints on domain models that *interfaces* have to be part of *component* and *components* cannot be directly interconnected and *components* must be interconnected through the *interface*. That is to say, as a modeling language in software architecture domain, M_{SA} describes all available domain modeling elements and imposed constraints on all domain models built based on M_{SA}.

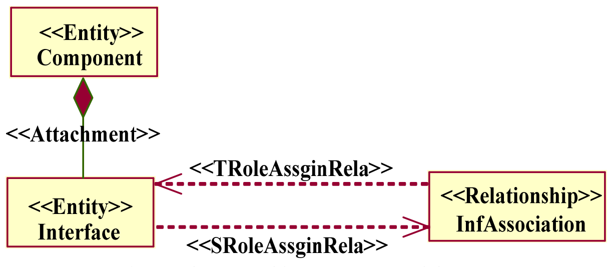


Fig.1 Software architecture metamodel MSA

A. *Structure Mapping of Metamodels*

Set of constant of name denoting name of modeling element of entity type has to be defined to constitute universe of discourse of interpretation of XMML, which is the basis of other structural mapping. There are three structural mapping rules as follows.

1. *Structural mapping rule* **STR1**(*name mapping of entity*): for every modeling element of entity metatype in the metamodel, the corresponding constant *Name* is generated according to its element name *Name*; the number of elements in constant set $C_{STR1}(M)$ generated by applying **STR1** on a metamodel M is the number of all entity elements in M, denoting $|STR_1(M)| = |EE|$ ($|EE|$ denotes the number of elements in set EE);

Subset of constant of name generated via M_{SA} is: $C_{STR1}(M_{SA}) = \{Component, Interface, InfAssociation\}$.

The following mapping rule is used to generate unary predicate statements via a metamodel to express which metatype modeling elements of entity type belong to.

2. *Structural mapping rule* **STR2**(*type mapping of entity*): for every modeling element of entity metatype named *Name* and belonging to type *P* in the metamodel, the corresponding unary predicate statement *P*(*Name*) is generated by applying *P* on *Name*; the number of elements in unary predicate statements set $L_{STR2}(M)$ generated by applying **STR2** on a metamodel M is the number of all entity elements in M, denoting $|STR_2(M)| = |EE|$;

Subset of unary predicate statements generated via M_{SA} is: $L_{STR2}(M_{SA}) = \{Entity(Component), Entity(Interface), Relationship(InfAssociation)\}$.

The following mapping rule is used to generate binary predicate statements via a metamodel to express the classification of modeling elements of association type.

3. *Structural mapping rule* **STR3**(*binary relationship mapping*): for every ordered pair of modeling element of entity metatype ($e1$, $e2$) connected by relationship R, in which name of element $e1$is *Name1* and name of element $e2$ is *Name2*, the corresponding binary predicate statement *R*(*Name*1, *Name*2) is generated by applying *R* on *Name*1 and *Name*2; the number of elements in binary predicate statements set $L_{STR3}(M)$ generated by applying **STR3** on a metamodel M is the number of all binary relationship in M, denoting $|STR_2(M)| = |RE|$. Assume that the edge connecting node is directed, the number of edges generated by connecting each other of n nodes may be $O(n^2)$, denoted $C_n^2 = \frac{n\times(n-1)}{2} = O(n^2)$, so $|RE| = O(|EE|^2)$.

Subset of binary predicate statements generated via M_{SA} is: $L_{STR3}(M_{SA}) = \{Attachment(Interface, Component), SRoleAssginRela(Interface, InfAssociation), TRoleAssginRela(InfAssociation, Interface)\}$.

It is obvious that the number of predicate statements generated by applying **STR1**, **STR2** and **STR3** on a metamodel M has a square relation with number of entity modeling elements, denoted

$$\sum_{i=1}^{3} STR_i(M) = |EE| + |RE| = O(|EE|^2)\cdot$$

Set of first-order logic statements generated by applying structural mapping on M is $L_{STR}(M) = L_{STR2}(M) \cup L_{STR3}(M)$, set of constants is $C_{STR}(M) = C_{STR1}(M)$.

B. *Semantic Mapping of Metamodels*

If distinction constraints between elements of the same type is not defined, then different modeling elements of the same entity metatype will be treated as equal in the natural deduction based on first-order logic, thus, equality comparison between elements would be meaningless, so it is necessary to establish semantic mapping rules for distinguishing elements. In addition, set composed of elements of the same type is limited and determined, if one element belongs to one metatype, then it must be an element of the set and cannot be out of set itself, so we must also create elements completeness rules.

1. *Semantic mapping rule* **MTR1**(*distinction of entity*): if $E_i = \{e_1, e_2, \ldots, e_n\}$ (Among them, n>1) is a set of all modeling elements of entity type belonging to the same entity metatype MT_i (MT_i is one of four entity metatypes defined by XMML) in the metamodel *M*, a group of formulas for distinguishing all elements of MT_i are generated by applying $e_j \neq e_k$ on any two different elements e_j and e_k in E_i to denote that elements of MT_i are different from each other, the number of which can be expressed as $C^2_{|E_i|} = \frac{|E_i| \times (|E_i| - 1)}{2}$. Set of all entity metatypes in which number of elements is greater than 1 in *M* can be denoted as $MT_M = \{MT_1, MT_2, \ldots, MT_k\}$, thus, *K* groups of formulas for different metatypes are generated by applying the above generating rules on all elements in MT_M, so the number of elements in set of formulas for distinguishing elements $F_{MTR1}(M)$ generated by applying **MTR1** on a metamodel *M* is $|MTR_1(M)| = \sum_{i=1}^{|MT_M|} (\frac{|E_i| \times (|E_i| - 1)}{2})$.

Subset of formulas for distinguishing elements generated by applying **MTR1** on all entity metatypes in which number of elements is greater than 1 in M_{SA} is: $F_{MTR1}(M_{SA}) = \{Component \neq Interface\}$.

2. *Semantic mapping rule* **MTR2**(*completeness of entity*): if $E_i = \{e_1, e_2, \ldots, e_n\}$ (Among them, n>1) is a set of all modeling elements of entity type belonging to the same entity metatype MT_i (MT_i is one of four entity metatypes defined by XMML) in the metamodel *M*, a formula of elements completeness of MT_i in the form of $\forall x.MT_i(x) \to (x = e_1) \vee (x = e_2) \vee \ldots \vee (x = e_n)$ is generated via MT_i and elements of set E_i belonging to MT_i to denote that elements of MT_i can only be in the set E_i. Set of all entity metatypes in *M* can be denoted as $MT_N = \{MT_1, MT_2, \ldots, MT_k\}$, thus, *K* formulas for different metatypes are generated by applying the above generating rules on all elements in MT_N, so the number of elements in set of formulas of elements completeness $F_{MTR2}(M)$ generated by applying **MTR2** on a metamodel *M* is the number of all metatypes of entity type in *M*, denoted $|MTR_2(M)| = |MT_N|$.

Subset of formulas of elements completeness generated by applying **MTR2** on all entity metatypes in M_{SA} is: $F_{MTR2}(M_{SA}) =$
$\{\forall x.Entity(x) \to (x = Component) \vee (x = Interface),$
$\forall x.Relationship(x) \to (x = InfAssociation)\}$

The number of first-order logic formulas generated by applying **MTR1** and **MTR2** on a metamodel *M* is:

$$\sum_{i=1}^{2} \mathrm{MTR}_i(M) = \sum_{i=1}^{|MT_M|} (\frac{|E_i| \times (|E_i| - 1)}{2}) + |MT_N| = O(|\mathrm{EE}|^2),$$

so the number of first-order logic formulas generated by applying all mapping rules on a metamodel *M* has a square relation with number of entity modeling elements, denoted $\sum_{i=1}^{3} \mathrm{STR}_i(M) + \sum_{i=1}^{2} \mathrm{MTR}_i(M) = O(|\mathrm{EE}|^2)$.

Semantic mapping rule **MTR2** is not necessary in most cases, so set of first-order logic formulas $T_L(M)$ corresponding to *M* is union of constant set $C_{STR}(M)$ and first-order logic statements set $L_{STR}(M)$ and formulas set for distinguishing elements $F_{MTR1}(M)$, denoted $T_L(M) = C_{STR}(M) \cup L_{STR}(M) \cup F_{MTR1}(M)$, among them, $C_{STR}(M)$ constitutes universe of discourse of interpretation of XMML, $L_{STR}(M)$ constitutes union of all subsets of interpretation of predicate symbols that is not empty, and $F_{MTR1}(M)$ constitutes constraints that elements in every subset of interpretation of unary predicate are different from each other.

The above mapping mechanism are our foundation for implementing automatic mapping tool used to map one metamodel to the corresponding first-order logic formulas set.

V. CASE STUDY

We illustrate the feasibility of our method for automatically verifying consistency of the metamodels based on formalization mapping rules by a metamodel M_{SS} shown in Fig. 2 as an example.

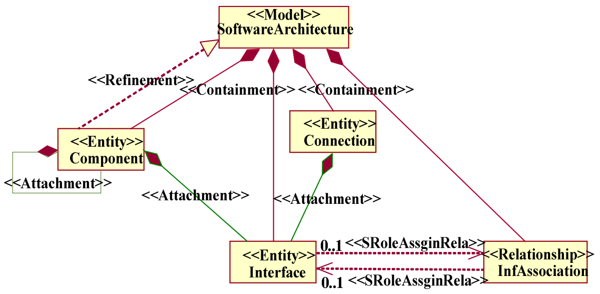


Fig.2 A metamodel M_{SS} of T_{SS}

The metamodel M_{SS} consists of modeling entities elements such as *softwarearchitecture*, *Component*, *Connection*, *Interface* and *InfAssociation* that means association between interfaces, it means that the model of *softwarearchitecture* contains entities such as *Component*, *Connection* , *Interface* and *InfAssociation*, and the entity *Component* can be refined into a new model of *softwarearchitecture*, and *Component* can be attached to itself, and it also establishes constraints on domain models that *interfaces* have to be part of *component* or *connection* and *components* or *connections* cannot be directly interconnected and *components* or *connections* must be interconnected through the *interface*.

We firstly build a small subset T_{SS} of T_X, and T_{SS} is a minimum set that can be used to determine whether M_{SS} satisfies XMML. T_{SS} contains three entity metatypes such as *Model*, *Entity* and *relationship* and five association metatypes such as *Containment*, *Attachment*, *Source Role assignment association, Target Role assignment association* and *refinement*. Symbol set and constraint formulas set of T_{SS} are listed as follows.

1. *predicate symbols set*
   $P_{SS}(L)$={*Model*(*x*),*Entity*(*x*),*relationship*(*x*),*Containment*(*x*,*y*),*Attachment*(*x*,*y*), *SRoleAssginRela*(*x*,*y*),*TRoleAssginRela*(*x*,*y*), *Refinement*(*x, y*)}
2. *Constraint formulas set* F_{SS}={

1) $\forall x,y.Containment(x,y) \rightarrow (Entity(x) \vee Relationship(x)) \wedge Model(y)$

   (F_{SS1}) *Containment* type constraint

2) $\forall x,y,z.Containment(x,y) \wedge Containment(x,z) \rightarrow (y=z)$

   (F_{SS2}) *Containment* uniqueness constraint

3) $\forall x,y.Attachment(x,y) \rightarrow Entity(x) \wedge Entity(y)$ (F_{SS3}) *Attachment* type constraint

4) $\forall x,y.Attachment(x,y) \rightarrow \neg Attachment(y,x)$ (F_{SS4}) *Attachment* no loop constraint

5) $\forall x,y,z.Attachment(x,y) \wedge Attachment(y,z) \wedge (x \neq y) \wedge (y \neq z) \wedge (x \neq z)$
   $\rightarrow AttaPath(x,y,z)$

   $\forall x,y,z.\neg AttaPath(x,y,z)$ (F_{SS5}) *Attachment* no two level path constraint

6) $\forall x,y.SRoleAssginRela(x,y) \rightarrow (Entity(x) \vee Relationship(x)) \wedge Relationship(x)$
   (F_{SS6}) *SRoleAssginRela* type constraint

7) $\forall x,y.SRoleAssginRela(x,y) \rightarrow \neg SRoleAssginRela(y,x)$
   (F_{SS7}) *SRoleAssginRela* no loop constraint

8) $\forall x,y.TRoleAssginRela(x,y) \rightarrow Relationship(x) \wedge (Entity(y) \vee Relationship(y))$
   (F_{SS8}) *TRoleAssginRela* type constraint

9) $\forall x,y.TRoleAssginRela(x,y) \rightarrow \neg TRoleAssginRela(y,x)$
   (F_{SS9}) *TRoleAssginRela* no loop constraint

10) $\forall x,y.SRoleAssginRela(x,y) \rightarrow \exists z.TRoleAssginRela(y,z)$
   $\forall x,y.TRoleAssginRela(x,y) \rightarrow \exists z.SRoleAssginRela(z,x)$
   (F_{SS10}) role assignment association appearing in pairs constraint

11) $\forall x,y.Refinement(x,y) \rightarrow Entity(x) \wedge Model(y)$ (F_{SS11}) *Refinement* type constraint

12) $\forall x,y,z.Refinement(x,y) \wedge Containment(x,z) \rightarrow (y=z)$
   (F_{SS12}) *Refinement* and *Containment* identity constraint}

A. Consistency Verification By Satisfaction Determination

Now we show how to manually verify metamodels consistency by determination rule of satisfaction relationship.

By related definition in [7], we can formalize M_{SS} built based on T_{SS} as an interpretation I_{SS} of T_{SS} consisting of the following three sets.

1. Universe of discourse of interpretation E_{SS}={*SoftwareArchitecture*,*Component*, *Connection*, *Interface*, *InfAssociation* };
2. Interpretation of all unary predicates in $P_{SS}(L)$:
(1) $Model^{I}$={*SoftwareArchitecture*}
(2) *EntityI* ={*Component, Connection, Interface*}
(3) $Relationship^{I}$= {*InfAssociation*}
3. Interpretation of all binary predicates in $P_{SS}(L)$:
(1) $Containment^{I}$={<*Component*, *SoftwareArchitecture*>,<*Connection*, *SoftwareArchitecture*>, <*Interface*, *SoftwareArchitecture*>,<*InfAssociation*, *SoftwareArchitecture*>},
(2) $Attachment^{I}$={<*Interface*, *Component*>,<*Interface*,*Connection*>, <*Component*, *Component*>}
(3) $SRoleAssginRela^{I}$ = {<*Interface,InfAssociation*>, <*InfAssociation, Interface*>}
(4) $Refinement^{I}$ = {<*Component*, *SoftwareArchitecture*>}

Can the metamodel M_{SS} in Figure 2 satisfy T_{SS}? Now we derive it by related definition in [6]. By $Containment^{I}$, It can be seen that type of contained four elements *Component*, *Connection, Interface* and *InfAssociation* at one end of *Containment* edge are *entity* or *Relationship* and type of containing element at the other end of *Containment* edge is *model*, so I_{SS} satisfies F_{SS1}, denoted $I_{SS} \models F_{SS1}$; similarly, we can derive $I_{SS} \models F_{SS2}$, $I_{SS} \models F_{SS3}$, $I_{SS} \models F_{SS5}$, $I_{SS} \models F_{SS6}$, $I_{SS} \models F_{SS8}$, $I_{SS} \models F_{SS9}$, $I_{SS} \models F_{SS11}$ and $I_{SS} \models F_{SS12}$. On the other hand, $I_{SS} \not\models F_{SS4}$ can be derived due to self-attached formed by one edge of <*Component*, *Component* >, and $I_{SS} \not\models F_{SS7}$ can be derived due to loop formed by two edges of <*Interface, InfAssociation*> and <*InfAssociation, Interface*>, and $I_{SS} \not\models F_{SS10}$ can be derived due to no target role assignment association edge corresponding to the edge <*Interface, InfAssociation*>, so we can derive that I_{SS} does not satisfy T_{SS}, denoted $I_{SS} \not\models T_{SS}$, so M_{SS} is not a legal instance of T_{SS}. By related definition in [7], we can derive that the metamodel M_{SS} is logical inconsistent.

B. Consistency Verification Based On Automatic Mapping

First-order predicate statements set $T_L(M_{SS})$ generated via metamodel M_{SS} based on automatic mapping rules for formalizing metamodels is built as follows.

Subset of constant of name generated by structural mapping rule **STR1** is: $C_{STR1}(M_{SS})$ = {*SoftwareArchitecture*,*Component*,*Connection*,*Interface*,*InfAssociation*}, with a total of 5 elements.

Subset of unary predicate statements generated by structural mapping rule **STR2** is: $L_{STR2}(M_{SS})$={*Model*(*SoftwareArchitecture*),*Entity*(*Component*),*Entity*(*Connection*), *Entity*(*Interface*),*Relationship*(*InfAssociation*)}, with a total of 5 elements.

Subset of binary predicate statements generated generated by structural mapping rule **STR3** is: $L_{STR3}(M_{SS})$={*Containment*(*Component*,*SoftwareArchitecture*), *Containment*(*Connection*,*SoftwareArchitecture*),*Containme*

$Containment(Interface,SoftwareArchitecture)$, $Containment(InfAssociation,SoftwareArchitecture)$, $Attachment(Interface,Component)$, $Attachment(Interface,Connection)$, $Attachment(Component,Component)$, $SRoleAssginRela(Interface,InfAssociation)$, $SRoleAssginRela(InfAssociation,Interface)$, $Refinement(Component,SoftwareArchitecture)\}$, with a total of 10 elements.

Subset of formulas for distinguishing elements generated by applying **MTR1** on all entity metatypes in which number of elements is greater than 1 in M_{SS} is: $F_{MTR1}(M_{SS})=\{Component \neq Connection$, $Component \neq Interface$, $Connection \neq Interface\}$, with a total of 3 elements .

So we can derive that $T_L(M_{SS})=C_{STR}(M_{SS}) \cup L_{STR2}(M_{SS}) \cup L_{STR3}(M_{SS}) \cup F_{MTR1}(M_{SS})$. By logical proof based on automatic theorem prover SPASS [19], we find that union of F_{SS} and $T_L(M_{SS})$ is logically contradictory, i.e. $F_{SS} \cup T_L(M_{SS}) \vdash False$.

Errors in the M_{SS} is corrected by validation results, including deletion of self-attached edge of <*Component*, *Component*>, adding of target role assignment association edge of <*InfAssociation, Interface*>, thus, we obtain the well-formed metamodel M_{SSC} shown in Fig. 3.

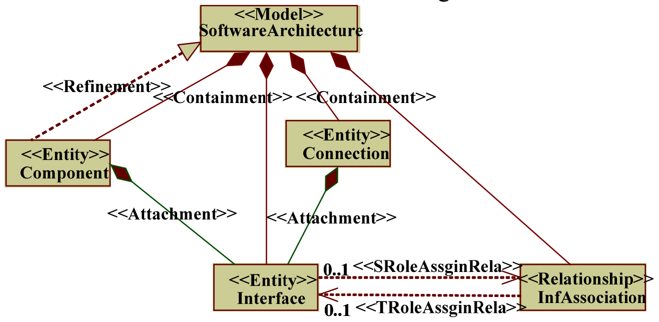


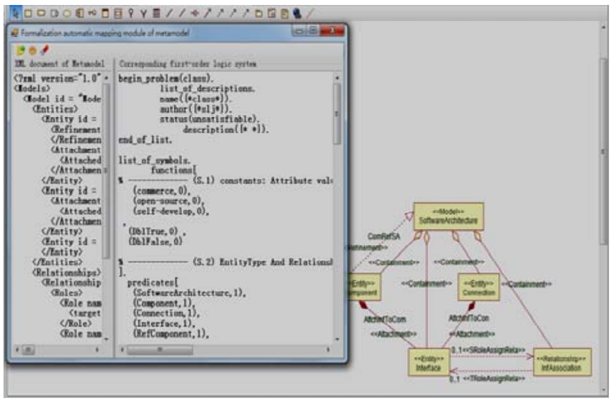
Fig.3 A well-formed metamodel M_{SSC} of T_{SS}

VI. DESIGN OF *MAPMMD*

Based on the above structure mapping rules and semantic mapping rules, we design our automatic mapping tool for formalizing metamodels called *MapMMD* (*Mapping of Metamodels Based on Meta-Domain*) to automatically translate metamodels defined on XMML concrete syntax scheme to the corresponding first-order predicate statements set in SPASS format.

Using .net 4.5 as development platform and using C# as programing language, we develop the new tool named *MapMMD* based on the improvement of the original system called *MapM* [6] and embed it in our modeling platform *Archware* [17] for building XMML metamodels and models, so *Archware* can verify characteristics of metamodels and models built based on XMML. Fig. 4 shows its running interface.

Below we briefly analyze the basic implementation mechanism of *MapMMD*. Firstly, XML document abstract syntax tree is generated via any metamodel based on XML document parsing interface provided by .net platform, and then, XML syntax tree is traversed using a series of XML syntax tree access interface based on .net to get element node attribute information and parent-child relationship information between element nodes, finally, through the analysis of a variety of information, the corresponding SPASS format constants, predicates and formulas can be combined and generated in accordance with various formalization mapping rules.



```
<?xml version="1.0"
<Models>
  <Model id = "Mode
    <Entities>
      <Entity id =
        <Refinement
        </Refinemen
        <Attachment
          <Attached
        </Attachmen
      </Entity>
      <Entity id =
        <Attachment
          <Attached
        </Attachmen
      </Entity>
      <Entity id =
      </Entity>
    </Entities>
    <Relationships>
      <Relationship
        <Roles>
          <Role nam
            <target
          </Role>
          <Role nam
```

```
begin_problem(class).
        list_of_descriptions.
        name({*class*}).
        author({*slj*}).
        status(unsatisfiable).
            description({* *}).
end_of_list.

list_of_symbols.
      functions[
% ------------ (S.1) constants: Attribute val
  (commerce,0),
  (open-source,0),
  (self-develop,0),
 ,
  (DblTrue,0) ,
  (DblFalse,0)

% ------------ (S.2) EntityType And Relationsl
].
  predicates[
  (SoftwareArchitecture,1),
  (Component,1),
  (Connection,1),
  (Interface,1),
  (RefComponent,1),
```

Fig. 4 Running interface of *MapMMD*

VII. EXPERİMENTS AND ANALYSİS

It is clear that the time $H_{Map}(M)$ it takes for translating any metamodel M using *MapMMD* is determined by traversal time $H_{Tvs}(M)$ of XML syntax tree generated via M. According to Structural mapping rule STR3, the number of edges generated has a square relation with number of nodes in the metamodel, so $H_{Tvs}(M)$ has also a square relation with number of nodes $N_{Tree}(M)$ in the tree, that is, $H_{Tvs}(M)=O(N_{Tree}(M)^2)$, and set of nodes corresponds to union of set of modeling elements of entity in the M, so we believe that $H_{Map}(M)$ has a square relation with number of entity elements $N_{Elm}(M)$ in the M, that is, $H_{Map}(M) =O(N_{Elm}(M)^2)$.

To validate the above result, we perform many experiments on automatic mapping for formalizing metamodels based on *MapMMD*. For example, we did the tests on software architecture metamodel M_{SSC} based on *MapMMD*, as a result, the XML document syntax tree code $X_{Tree}(M_{SSC})$ generated via XML parsing interface is translated into the corresponding first-order logic formulas in SPASS format $T_L(M_{SS})$ via *MapMMD*, and it only takes 0.05 second to finish translating on M_{SSC} that includes 5 entity elements. Diagram of relationship between translating time and number of entity elements in the metamodel is shown in Fig. 5. From Fig. 5, We can get the same result as the above analysis.

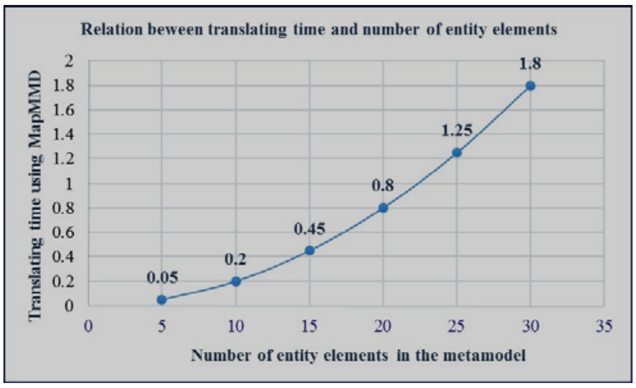


Fig. 5 Relationship between translating time and number of entity elements

So we conclude that size of the metamodel which can be translated using *MapMMD* is not limited, and the time it takes for translating using *MapMMD* has a square relation with size of the metamodel.

VIII. CONCLUSION

Most DSMMLs are not defined in formal way, it makes it difficult to precisely represent its semantics and to automatically analyze metamodels's characteristics. In response, the paper proposes a new approach for automatically verifying metamodels consistency by an automatic mapping mechanism based on formalization of DSMML named XMML, and then we give a typical case, based on this, we design corresponding automatic mapping tool and perform experiments and analyze experiments data. Experiments results show that our method is feasible and effective.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGMENT

The authors are grateful to Prof. Hua Zhou and Dr. Xinping Sun for valuable discussions. This work is supported by the National Natural Science Foundation of China (No. 61363022).

REFERENCES

[1] Steven K, Juhz-Pekka T, Domain-specific modeling: Enabling full code generation New Jersey, USA: John Wiley & Sons, 2008.
[2] Jouault F, and B´Ezivin J, "Km3: A dsl for metamodel specification", In FMOODS pp. 171–185.2006
[3] Vanderbilt University, GME User's Manual, 2013, http://www.isis.vanderbilt.edu/Projects/gme/.
[4] Object Management Group, Unified Modeling Language: Superstructure version 2.4.1, 2011, http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF.
[5] Object Management Group, Meta Object Facility Specification version 2.4.1, 2011, http://www.omg.org/spec/MOF/2.4.1.
[6] Tao Jiang, "Formalizing Domain-Specific Metamodeling Language XMML Based on First-order Logic", Journal of Software, . 7(6): pp. 1321-1328.2012
[7] Tao Jiang, "Research on Metamodel Consistency Verification Based on First-order Logical Inference", Electrical Review, 88(1b): pp. 132-136, 2012.
[8] Shan L, Zhu H, "A formal descriptive semantics of UML", in Proceedings of the 10th International Conference on Formal Methods and Software Engineering, Berlin, Germany, Springer-Verlag, pp. 375-396, 2008.
[9] Hao Fei, Dong Qingchao, Zeng Guangju, "A Method of UML Models Verification Based on Description Logic", Computer and Digital Engineering, (11):58-62.2011
[10] Snook, C. and Butler, M., "UML-B: Formal Modeling and Design Aided by UML", ACM Transactions on Software Engineering and Methodology. 15(1): p. 92–122, 2006.
[11] Moller, M., et al., "Linking CSP-OZ with UML and Java: A Case Study", in Integrated Formal Methods, Springer Berlin/ Heidelberg. p. 267-286, 2004.
[12] MetaCase, MetaEdit+ Version 5.0 User's Guide, 2014, http://www.metacase.com/support/50/manuals/meplus/Mp.html.
[13] Matthias Tichy, Christian Krause, and Grischa Liebel, "Detecting Performance Bad Smells for Henshin Model Transformations", in Proceedings of the Second Workshop on the Analysis of Model Transformations of 14th International Conference on Model Driven Engineering Languages and Systems (AMT 2013), Miami, FL, USA, September 29, 2013, ACM/IEEE, 2013.
[14] Bernhard Rumpe and Ingo Weisem¨oller, "A Domain Specific Transformation Language", in Proceedings of International Workshop on Models and Evolution of 14th International Conference on Model Driven Engineering Languages and Systems (ME 2011), Wellington, New Zealand, October 16-21, ACM/IEEE, 2011.
[15] Jackson.E.K, Sztipanovits.J, "Formalizing the Structural Semantics of Domain-Specific Modeling Languages", Journal of Software and Systems Modeling, 2009, **8**(4): pp. 451-478.
[16] Jackson.E.K, Tihamer Levendovszky, and Daniel Balasubramanian, "Reasoning about Metamodeling with Formal Specifications and Automatic Proofs", in Proceedings of 14th International Conference on Model Driven Engineering Languages and Systems (ME 2011), Wellington, New Zealand, October 16-21, ACM/IEEE, 2011.
[17] Sun XP, A Research of Visual Domain-Specific Meta-Modeling Language and Its Instantiation, Kunming: Yunnan University.2011.
[18] Cheng MZ, Yu JW, Logic foundation—first-order logic and first-order theory, Chinese People University Press, Beijing, 2003.
[19] Max-Planck-Institut Informatik, SPASS Tutorial, 2012, http://www.spass-prover.org/tutorial.html.