Comput Syst Sci & Eng (2018) 5: 299–308

International Journal of
Computer Systems
Science & Engineering

A description method for formalizing domain-specific modelling language

Tao Jiang

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

Many Domain-Specific Modelling Languages (DSML) can not formally define their semantics, which inevitably brings many problems, such as accurate description and automatic verification of model properties. In this paper, we propose a formal description method of the structural semantics of DSML for verifying consistency of models built based on DSML. Firstly, domain indicating structural semantics of DSML is formally defined based on algebra, and then, we briefly describe our framework for formalizing DSML and verifying consistency of DSML and its models and use a classic case to illustrate our approach; based on this, we construct an automatic translator for formalizing DSML and its models. Finally, many successful experiments on automatic translating and automatic verifying show feasibility of our formal approach

Keywords: Domain-Specific Modelling Language (DSML), domain, formal framework, automatic verification

1. INTRODUCTION

As one of essential components of model-driven architecture (MDA) [1], DSML is a modelling language for specific domain system modelling (DSM) [2] and can be used to build specific domain models, and it is an instance built based on Domain-Specific Metamodeling Languages. We describe DSML's abstract grammar and structural semantics by one metamodel that is a special model used for describing the constraints that all models built based on DSML must adhere to. That is to say, the metamodel is a semantic-based representation of Domain-Specific Modelling Languages.

In the paper, regardless of behavioural semantics that concern dynamic features during execution, we only consider structural semantics of DSML that describe static structural constraints imposed on relationship between model elements [3].

To be fair to say, DSML differs from any traditional programming language that uses an eBNF to define its syntax [4] and has a type-system. The first difference between the two is that DSMLs use metamodels to define their specification. Metamodels characterize rich layered architecture and various semantic relations between modelling entities by using a model similar to UML class diagram. Different from the BNF syntax, metamodels regard all modelling entities as same level concepts. The second difference lies in that we can apply DSMLs in varied ways such as model transformations used to translate from domain-specific syntaxes to different formal languages, and analysis and reasoning on models properties and so on.

There are some important aspects for formalizing DSML. First, there must be an algebraic definition for structure of DSML. Second, we can obtain formal specifications which are independent on tools from tool-dependent models. Third, we can analyze and automatically reason about many properties of DSML and its models such as consistency based on the formal foundation. But up to now, these problems have not yet been well solved. We illustrate it with the following several instances. VMP [5] is a visual, precise and multilevel metamodeling framework, but It can't express constraint rules bound to the language itself very well. The mature metamodelling Environment (GME) [6] can express rich constraints, but its formal specifications is not independent of implementation code of complex tools. Standards such as the UML superstructure [7] and Meta-Object Facility (MOF) [8] define the syntax and process via a group of metamodels and express semantics by us-

ing expanded Object Constraint Language [9], so it is difficult to precisely analyze and automatically reason about UML models.

Different from those top-down approach using the meta-metamodel to drive metamodels that drives models. We use a bottom-up method to formalize structural semantics of DSML. We start with models and ends with metamodeling through modelling. That is to say, first, we define a simple mathematical structure called a domain to describe the rich syntaxes and structural semantics of DSML. Next, we define one mapping that map one domain to the first-order logic formulas set, which is called domain mapping. Third, we redefine domain using the same basic mathematical structure based on first-order logic and propose an analysis method on domain properties using logical reasoning and illustrate it by a classic case. Our formal approach is flexible enough to adapt to not only DSML named XMML (XML-based Domain-Specific metamodelling language) designed and developed by ourselves [10] but also other DSMLs.

In order to fully apply our formal method, we construct an automatic translator for formalizing DSML and its model called LTtranslMSS (Logical translator on DSML and models Based on structural semantics) to perform translation of XML format metamodels or models built based on XMML. Finally, we execute automatic mapping experiment on DSML and its models using LTtranslMSS to illustrate the practicability of our formal method, and perform automatic verification test to illustrate automatic theorem prover's restrictions on size of the metamodel and its models.

2. RELATED WORKS

Let's take a look at the UML community first. In Shan L's paper [11], she defines the modelling language by a metamodel in UML class diagram and characterizes the metamodel's semantics via the mapping from metamodels to corresponding sets of first-order logic formulas over the first order languages and implement a mapping tool called LAMBDES to translate metamodels and models into first order logic formulas. There are several differences between Our approach and Shan L's. Our XMML is mainly used to build Domain-Specific models such as software architecture, network topology and so on, but Shan L's focuses on UML class diagram models, state machine models and other UML models. Because core modelling elements and main relationship between elements contained in the two languages are different, the former includes entity and the relationship such as refinement, containment and so on, the latter consist of elements such as classes, generalization, realization and dependency and so on, their concrete syntax and structural semantics have to be different, which inevitably leads to the different formalizing method and translating rules and automatically reasoning mechanisms. Next, our formalization of XMML involves metamodelling language layer, and Shan L's focuses on modelling language layer, so abstraction level of the two is different.

Mustafa Al-Lail et al propose an approach for directly analyzing UML Class Models' temporal properties expressed in TOCL and show the feasibility of the approach via an example based on the Steam Boiler Control System [12]. Without considering formalization of class metamodel, the approach can not achieve precise and systematic model validation.

Ruzhen Dong et al propose a formal model-driven engineering method rCOS for formally modelling software architectures to bridge the gap between formal techniques and their latent support for developing practical software. The focus of the approach is composition and refinements of components rather than the automated reasoning of models [13].

Look at DSM community again. Faiez Zalila et al use a language called TOCL to formally express system requirements and interpret verification results in order that system designers need not learn more formal method. They have integrated the verification tools on a DSML in order to assist developer verifying many properties on executable models. Different from our method, it can not realize automated reasoning because there is no automatic mapping mechanism [14].

Ethan K. Jackson and Wolfram Schulte design a new specification language based on model theory and develop the FORMULA program for dealing with search problems and recursive definitions on basis of automated formal analysis via constraint solving [15]. As a novel formal specification language for efficient reasoning and fast verifying based on open-world logic programs and behavioral types, FORMULA is based on Horn logic [16], rather than first-order logic used in our method.

But using our method, we can implement automatically reasoning on consistency of DSML and its models by automatically translate XML format DSML and its models into the corresponding first-order logic system based on our automatic translating mechanism for formalizing DSML. This is very important for formalizing DSML and verifying models.

3. DOMAIN

In my another published paper, we have created a mathematical description of domain structure based on algebra and discussed domain emptiness, domain equivalence and domain mapping mechanism. Here, we only give the brief description of domain, and its details can be seen in [17], based on this, we discuss properties of domain mapping and establish a definition of DSML.

A domain R consists of 5 sets such as domain signature set Sindicating name of modelling element, constraint signature set S_c as an extended set of S, alphabet set Ω indicating name of model element, constant set O indicating domain attributes and constraints formulas set C denoting a group of well-formedness rules on domain models.

Representing structural semantics of the DSML using domain can unify DSML and its models built based on it in the domain so that we can discuss domain properties better.

After defining domain structure and discussing domain properties based on algebra, we cannot reason about consistency of DSML and its models, so we have to create a translating mechanism called domain mapping to translate one domain into first-order logic system, and its details can be seen in [17]. Properties of domain mapping are discussed in the following.

Theorem 1 *Domain mapping δ is a one-to-one and onto function from domain $R = < S, S_c, \Omega, O, C >$ to first-order logic system T_R.*

Proof Let x is an arbitrary domain. The mapping δ consists of several identical mappings or equivalence mappings, so domain x must return to itself by the mapping δ and then the inverse

mapping δ^{-1}, that is, $\delta^{-1}(\delta(x))=x$, to $\delta(x)=\delta(y)$, after adding δ^{-1} on both sides of $=$, we can get $\delta^{-1}(\delta(x))=\delta^{-1}(\delta(y))$, thus $x = y$,so δ is one-to-one function. Additionally, there must be a domain $x$that makes $\delta(x) = T_R$ for any first-order logic system T_R. Thus, M is one-to-one and onto function, so we can conclude that domain mapping δ and its inverse mapping $\delta^{-1}$are both bijections.

After domain mapping have been proved to be a bijection, we can reason about domain consistency via T_R. On the basis of the domain, we give the formal definition of DSML as follows.

Definition 1 (DSML). *A DSML $L_R$is 2-tuple consisting of domain $R =< S, S_c, \Omega, O, C >$ and its domain mapping H*, that is, $L_R =< R, H >$.

4. FORMALIZING DSML AND ITS MODELS

4.1 A Formal Framework of DSML

We use a metamodel to represent DSML's structural semantics, thus, once having completed formalization on a metamodel, we complete formalization on DSML.

According to the definition about domain, we add symbols sets consisting of symbols signature set S, constraint signature set Sc and constant set O and constraints formulas set C into first-order logic formalized system called Q predicate calculus [18], thus, we establish a metamodel formalized system based on first-order logic called $T_Q(M)$ (M denotes one metamodel). According to first order logic semantic theory, any model instance created using metamodel M can be seen an interpretation of formalized system $T_Q(M)$, and a collection of all entity type elements in the instance can be regarded as universe of discourse of interpretation, so power set of the term algebra $T_S(\Omega \cup O)$ over S generated by $\Omega \cup O$ can be equivalent to set of all models M_S built based on M, denoted $M_S = P(T_S(\Omega \cup O))$, where P means power set of set. At the same time, M_S is also a collection of all possible interpretations of $T_Q(M)$. Based on this, we can use determination rule of satisfaction relationship to verify well-formedness of any model as an instance of metamodel M.

uniqueness of XMML makes it possible for us to formalize XMML by artificial derivation and logical proving. But metamodels built on basis of XMML are many and varied, similarly, models built on basis of DSML are also many and varied, thus it is impossible for us to perform a manual derivation for every metamodel or each model, which inevitably lead to full artificial reasoning on properties of metamodels or models, so an automatic mapping mechanism for formalizing any metamodel and its models has to be created. With improvement to UML mapping mechanism in the literature [11], and combined with first-order logic theory, we establish a framework for formalizing any metamodel and its models.

To build a metamodel formalized system, the key is to establish a first-order language symbol set and a group of constraint formulas based on symbol set. We can derive a first-order language symbol set $\sum$ from abstract syntax of a metamodel, which consists of a set of predicate symbols, so the mapping from metamodels to $\sum$ called symbol mapping L has to be firstly established. Set of constraint formulas can be derived from structural semantics of a metamodel, which consists of uniqueness of classification of entity type elements, type constraints and multiple constraints of association type elements and so on, thus we then establish the formula mapping $C_{\sum}$ based on $\sum$. Upon completion of formalizing any metamodel, we can analyse logical consistency of itself. In addition, to verify consistency of models built based on any metamodel, the mapping from a model to a corresponding first-order logic statements set based on $\sum$ called model mapping $S_{\sum}$ has to be established to determine whether a model as an interpretation of a metamodel formalized system can satisfy a metamodel. Thus, a framework for formalizing any metamodel consists of symbol mapping L, formula mapping $C_{\sum}$, model mapping $S_{\sum}$, metamodel consistency verification and determination of satisfaction of a model to a metamodel and so on. The architecture of the framework is shown in Figure 1.

First-order language symbol set $\sum(M)$ generated via symbol mapping is a union set of constant symbol set $C_{\sum}(M)$ and predicate symbol set $P_{\sum}(M)$, that is, $\sum(M) = C_{\sum}(M) \cup P_{\sum}(M)$. Constraint formulas set $A_{\sum}(M)$ based on $\sum(M)$ generated via formula mapping contains the following three sets: typed constraints set $A_{CU}(M)$ used to define classification completeness and uniqueness of entity type elements, type constraints set $A_T(M)$ used to establish type constraints of entity elements located on both sides of the association and multiple constraints set $A_M(M)$ used to describe numbers of entity elements that are allowed at two endpoints connected by an association edge, so $A_{\sum}(M) = A_{CU}(M) \cup A_T(M) \cup A_M(M)$.

Based on the framework, we can establish corresponding automatic mapping rules for finishing symbol mapping, formula mapping and model mapping, currently, these rules can only support translation of XML format metamodels or models built based on XMML, without supporting other DSMLs, and its details can be seen in my another paper [19].

4.2 Consistency of Metamodel and its Models

We find it very difficult to find an satisfiable interpretation for constraint axiom set $A_{\sum}(M)$ of any metamodel formalized system $T_Q(M)$ manually, so we can only use automatic theorem prover to automatically prove logical consistency of constraint formula sets $A_{\sum}(M)$. logical consistency of metamodel has been defined in my another paper [19].

After metamodel formalized system called $T_Q(M)$ is proved to logically consistent, metamodel M has to exist a satisfiable interpretation, that is to say, there are legal models built based on metamodel M, thus we can analyse many models' properties such as consistency in the domain represented by metamodel M. At the point of first-order logic [18], a legal model S as an instance of metamodel M is an interpretation that satisfies all constraint formulas of $A_{\sum}(M)$ of metamodel formalized system $T_Q(M)$. Similarly, an illegal model S is an interpretation that cannot satisfies $A_{\sum}(M)$. An interpretation of $T_Q(M)$, satisfaction relationship of the interpretation of $T_Q(M)$ to $T_Q(M)$ and logical consistency of model are formally defined in [19]. According to equivalence relationship of satisfaction and logical consistency, reference to the literature [11], we can deduce determination method of model consistency. Please refer to the

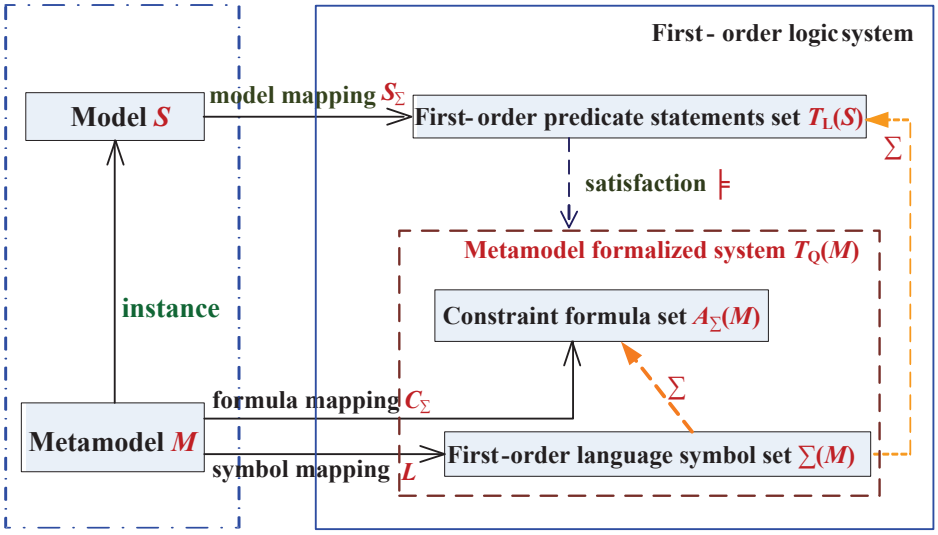

Figure 1 A formal framework of metamodel.

literature [19].

4.3 Case Study

We illustrate our formal method by using software architecture metamodel W_{SA} shown in Figure 2 as an example.

The metamodel W_{SA} consists of modeling elements of entity type such as *SoftwareArchitecture*, *Component*, *Connection*, *Interface* and modeling elements of association type such as *AttchInfToCom* and*AttchInfToCon* denoting attachment relationship, *ComRefSA* denoting refinement relationship and*InfAssociation* denoting association relationship, it denotes that software architecture consists of component and connection, and also builds constraint rules on all models in the domain that *interfaces* have to be a part of *component* or *connection* and *components* or *connections* cannot be directly interconnected and *components* or *connections* must be interconnected through the *interface* and any component can be refined into a new software architecture model.

According to symbol mapping rules, constant symbol set generated via W_{SA} is empty, that is, $C_\Sigma(W_{SA}) = \emptyset$, predicate symbol set generated via W_{SA} is $P_\Sigma(W_{SA}) = \{SoftwareArchitecture(x)$, $Component(x)$, $Connection(x)$,$Interface(x)$, $AttachInfToCom(x,y)$, $Attach InfToCon(x,y)$, $InfAssociation(x,y)$,$ComRefSA(x,y)$, $Software ArchitectureContainment(x,y)\}$.

Typed constraints set generated by applying formula mapping rules on W_{SA} is: $A_{CU}(W_{SA}) = \{$

$\forall x.SoftwareArchitecture(x)?Component(x)?Connection(x)?Interface(x)$,

$\forall x.SoftwareArchitecture(x) \rightarrow \neg Component(x)$,

$\forall x.SoftwareArchitecture(x) \rightarrow \neg Interface(x)$,

...

$\forall x.\ Connection(x) \rightarrow \neg Interface(x)\}$, with a total of 7 formulas; and type constraints set generated via W_{SA} is: $A_T(W_{SA}) = \{$

$\forall x,y.\ AttachInfToCom(x,\ y) \rightarrow Interface(x) \wedge Component(y)$

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

$\forall x,y.\ \ ComRefSA(x,y) \rightarrow Component(x) \wedge Software Architecture(y)$

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

$\forall x,y.InfAssociation(x,y) \rightarrow Interface(x) \wedge Interface(y)$

$\}$, with a total of 7 formulas; and multiple constraints set generated via $W_{SA}$is: $A_M(W_{SA}) = \{$

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

$\forall x,y,z.\ InfAssociation(x,y) \wedge InfAssociation(x,z) \rightarrow (y = z)\}$,

with a total of 2 formulas. Thus, Constraint formula set generated via W_{SA} is union of above three sets, that is

$A_\Sigma(W_{SA}) = A_{CU}(W_{SA}) \cup A_T(W_{SA}) \cup A_M(W_{SA})$.

After we add$\sum(W_{SA})$ and $A_\Sigma(W_{SA})$ into predicate calculus Q to form metamodel formalized system $T_Q(W_{SA})$, formalization of W_{SA}based on first-order logic is finished. After $A_\Sigma(W_{SA})$ have been verified based on automatic theorem prover, there is no contradiction in it, and we can conclude that W_{SA} is logically consistent in the domain, there exist an interpretation that can satisfy metamodel W_{SA}, so it makes sense to perform consistency verification of models built based on W_{SA} in the domain.

Based on formalization of DSML, we discuss our method of models' consistency verification by using two models in software architecture domain named $S_1(W_{SA})$ and $S_2(W_{SA})$ shown in Figure 3 and Figure 4 as examples. Both $S_1(W_{SA})$ and $S_2(W_{SA})$ describe a client/server two-tier architecture connected by middleware. In addition, some elements such as $P1$, $P2$and so on in Figure 4 indicate interfaces attached to a component or a connection.

According to model mapping, a group of first-order predicate statements corresponding to $S_1(W_{SA})$ generated by applying model mapping rules on $S_1(W_{SA})$ is: $T_L(S_1) = \{Component(ClientA)(S_{11})$,$Component(ClientB)(S_{12})$,$Component(Server)\ (S_{13})$,

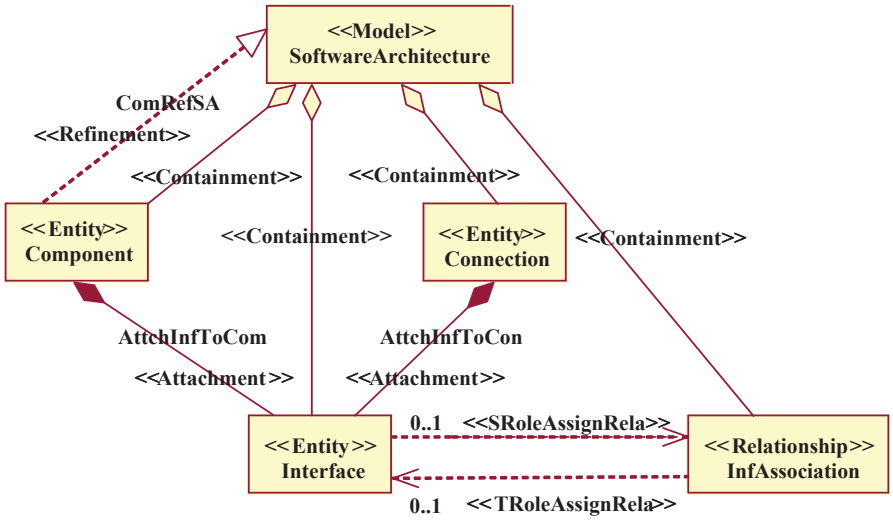

Figure 2 Software architecture metamodel W_{SA}.

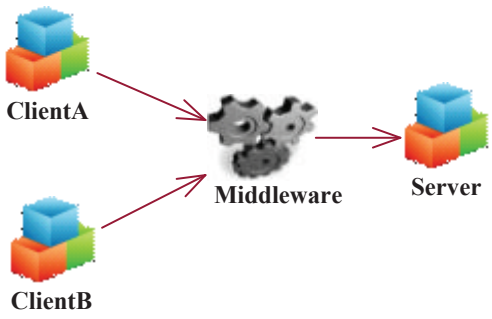

Figure 3 Model $S_1(W_{SA})$ built based on W_{SA}.

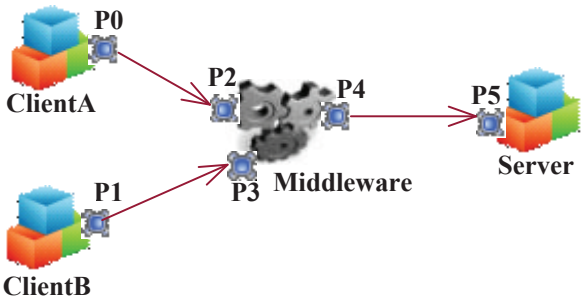

Figure 4 Model $S_2(W_{SA})$ built based on W_{SA}.

Connection(*Middleware*) (S_{14}),*InfAssociation* (*ClientA*, *Middleware*) (S_{15}),*InfAssociation*(*ClientB*, *Middleware*) (S_{16}),*InfAssociation* (*Middleware*, *Server*) (S_{17})}. Union of $A_{\sum}(W_{SA})$ and $T_L(S_1)$ is contradictory based on automatic theorem prover, that is, $A_{\sum}(W_{SA}) \cup T_L(S_1) \vdash False$, and we can find that contradiction is generated by violating association type constraint formula in $A_T(W_{SA})$ $\forall x, y$.InfAssociation$(x, y) \rightarrow$ Interface$(x) \wedge$Interface(y) ,thus we can determine that $S_1(W_{SA})$ is not consistent in the domain W_{SA} by Inference 1.

Similarly, a group of first-order predicate statements corresponding to $S_2(W_{SA})$ generated by applying model mapping rules on $S_2(W_{SA})$ is: $T_L(S_2)$= {*Component*(*ClientA*), *Component*(*ClientB*), *Component*(*Server*), *Connection*(*Middleware*), *Interface*($P0$), *Interface*($P1$), *Interface*($P2$), *Interface*($P3$), *Interface*($P4$), *Interface*($P5$), *AttachInfToCom*($P0$,*ClientA*), *AttachInfToCom*($P1$,*ClientB*), *AttachInfToCon*($P2$,*Middleware*), *AttachInfToCon*($P3$,*Middleware*),*AttachInfToCon*($P4$,*Middleware*), *AttachInfToCom*($P5$,*Server*), *InfAssociation*($P0$,$P2$), *InfAssociation*($P1$,$P3$), *InfAssociation*($P4$, $P5$)}. Union of $A_{\sum}(W_{SA})$ and $T_L(S_2)$ is logically consistent based on automatic theorem prover, that is, $A_{\sum}(W_{SA}) \cup T_L(S_2) \vdash True$, thus we can determine that $S_2(W_{SA})$ is consistent in the domain W_{SA}.

5. DEVELOP OF LTTRANSLMSS

Based on symbol mapping, formula mapping and model mapping, through the improvement and expansion of the original system LTtranslMSS in my another paper [19], we design an automatic mapping tool for formalizing DSML and its models called LTtranslMSS to automatically translate XML format metamodels and models defined based on XMML grammar style to the SPASS format predicate statements set [20], so we can automatically verify their many properties such as consistency.

LTtranslMSS is made up of automatic translating subsystem that is used to formalize metamodel called *TranslMBD* (*Translating of Metamodel Based on Domain*) and translating mapping subsystem that is used to formalize model called *TranslSBD* (*Translating of inStances Based on Domain*). *TranslMBD* implements formalization of metamodel via symbol mapping and formula mapping to finish verification of logical consistency of $T_Q(M)$. *TranslSBD* implements formalization of model via model mapping based on formalization of metamodel to finish verification of logical consistency of model. *TranslMBD* will generate metamodel logic system, and*TranslSBD* will generate model logic system, and both have to be merged into one logic system named domain logic system as input of SPASS.Logical architecture of*TranslMBD* and*TranslSBD* is shown in Figure 5.

Through improving the previous version named LMapMSS, we construct an automatic translator for formalizing DSML and its models named LTtranslMSS by using Visual C# as development tool on windows platform, and then integrate it into our visual metamodelling and modelling tool *Archware*. Thus, *Archware* becomes a complete platform including visual modelling, automatically translating of metamodel and its models and automatically reasoning on models.

6. EXPERIMENTS AND ANALYSIS

6.1 Experiments on Automatic Translating

To verify the availability of the LTtranslMSS, we do many experiments for automatically translating metamodels by using *TranslMBD*. we conclude that there is a square relation between cost of translation time on one metamodel M and total number of entity modeling elements contained in the M according to square relation between traversal time on M syntax tree and number of nodes in M.

we also do many tests to verify whether there is a square relation between the two by using *TranslMBD*. For instance, we executed automatically translating on metamodel W_{SA} in 4.3 section by using *TranslMBD*. As a result, syntax tree code using XML format $X_{Tree}(W_{SA})$ generated via XML parsing interface is listed in Appendix A, and the SPASS format first-order logic code $T_Q(W_{SA})$ created via translating interface is listed in Appendix B. Because W_{SA} contains only 5 entity elements, we only spend 0.06 second to complete the translation on W_{SA}. In addition, after we perform many automatically translating on different sizes of metamodel, such as the metamodel that includes 10, 15 or 20 entity elements and so on, we find that translation time grows by square relation with the increase of the number of elements contained in the W_{SA}. Relationship between the two is shown in Figure 6.

We can come to the conclusion that size of the metamodel that we can use*TranslMBD* to translate is unrestricted, and translation time grows by square relation with the increase of the metamodel size. The above conclusion can be equally applied in the automatic translating of models using *TranslSBD*.

6.2 Experiments on Automatic Verification

According to research results of my another published paper [21], there is a square relation between the total number of first-order logic formulas generated by formalizing any metamodel and number of entity modeling elements contained in the metamodel; in addition, the time spent on completion of proof has an exponential relationship with the number of logic formulas. So as entity elements of any metamodel increase, the corresponding logic formulas grow in square relation, which will lead to an exponential increase in time spent for proving based on SPASS.

We do a lot of experiments on automatic verification of metamodels and its models based on SPASS and other automatic theorem prover such as Otter [22]. We find through experiment that the upper limit of the number of formulas generated via any metamodel is about 1000. In particular, if a metamodel contains less than 1000 formulas, SPASS or Otter only needs less than 10 seconds to complete proof; otherwise, it takes more than 10 seconds for SPASS or Otter to complete proof. On the other hand, whether the metamodel contains contradictions is another important factor to determine the proof time length. To any metamodel with contradiction, proof time is relatively short, usually not more than a few seconds, but if the metamodel contains no contradictions, SPASS or Otter have to spend much longer time to finish proof, sometimes it cannot normally stop running.

We also execute automatic verification experiments on the constraint formula set $A_{\sum}(W_{SA})$ generated by metamodel W_{SA}shown in Figure 5 based on SPASS. There are only 21 formulas contained in the $W_{SA}$and there is no contradiction in it,so it only takes 1 seconds to finish proof on $A_{\sum}(W_{SA})$. We also finish consistency verification of model $S_1(W_{SA})$ and $S_2(W_{SA})$ built based on W_{SA}. We find that it only takes less than 1 second to finish proof on union of $A_{\sum}(W_{SA})$ and$T_L(S_1)$ generated by$S_1(W_{SA})$ due to contradictions contained in $T_L(S_1)$, but SPASS spends 2 seconds to complete proof on union of $A_{\sum}(W_{SA})$ and$T_L(S_2)$ generated by$S_2(W_{SA})$ since $T_L(S_2)$ is logically consistent.

We can come to the conclusion that the number of formulas generated via any metamodel should not exceed 1000, or we have to spend much longer time to prove or cannot even terminate running of SPASS due to undecidability of first-order logic, and the above conclusion can be equally applied in the automatic verification of models.

7. CONCLUSION

Many Domain-Specific Modelling Languages (DSML) can not formally define their semantics, which inevitably brings many problems, such as accurate description and automatic verification of model properties. In order to solve this problem, we

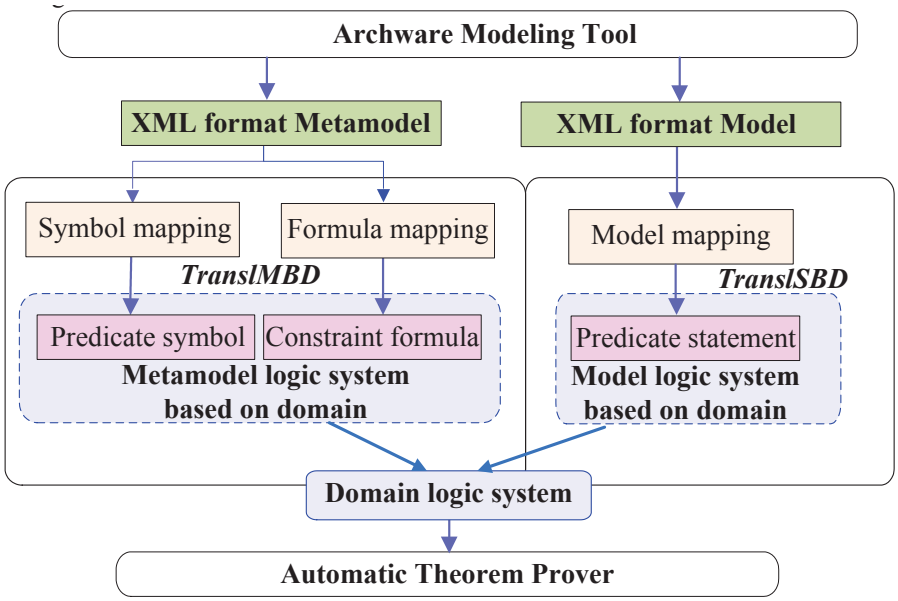

Figure 5 Logical architecture of *TranslMBD* and*TranslSBD*.

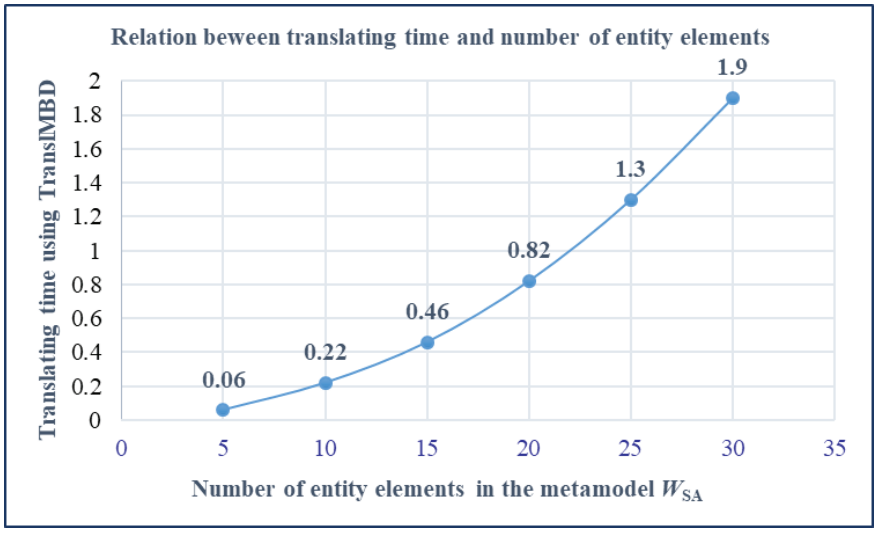

Figure 6 Square relationship between translating time and number of entity elements.

present a formal description method of DSML's structural semantics for verifying models' consistency to unify DSML and its models in the same domain. Based on this, we create our framework for formalizing DSML and verifying consistency of DSML and its models and illustrate it by a classic case. Finally, we construct an automatic translator for formalizing DSML and its models and do many experiments on automatic translating and automatic verifying to confirm the practicability of our formal approach.

Acknowledgements

The authors are grateful to Prof. Hua Zhou, Dr. Xinping Sun and Dr. Yong Yu for valuable discussions. This work was supported by the National Natural Science Foundation of China under Grant No. 61363022.

Appendix A XML document syntax tree of $W_{SA} - X_{\text{Tree}}(W_{SA})$

```
<?xml version="1.0" encoding="utf-8" ?>
  <Models>
  <Model id = "Model-1" name ="SoftwareArchitecture" kind ="SoftwareArchitecture">
  <Entities>
  <Entity id = "Entity-2" name = "Component" kind ="Component">
  <Refinement id="Refinement-3" name="ERefinedModel" kind="ERefinedModel" refinedmodel="Model-1">
  </Refinement>
  <Attachment id="Attachment-4" name="AttachInfToCom" kind="AttachInfToCom">
  <AttachedEntity AttachedEntityID="Entity-8" ></AttachedEntity>
  </Attachment>
  </Entity>
  <Entity id = "Entity-6" name = "Connection" kind ="Connection">
  <Attachment id="Attachment-7" name="AttachInfToCon" kind="AttachInfToCon">
  <AttachedEntity AttachedEntityID="Entity-8" ></AttachedEntity>
  </Attachment>
  </Entity>
  <Entity id = "Entity-8" name = "Interface" kind ="Interface">
  </Entity>
  </Entities>
  <Relationships>
  <Relationship id = "Relationship-9" name = "InfAssociation" kind ="InfAssociation">
  <Roles>
  <Role name = "SRoleAssginRela" multilower="0" multiupper="1">
  <targetitem itemid = "Entity-8"></targetitem>
  </Role>
  <Role name = "TRoleAssginRela" multilower="0" multiupper="1">
  <targetitem itemid = "Entity-8"></targetitem>
  </Role>
  </Roles>
  </Relationship>
  </Relationships>
  </Model>
  </Models>
```

Appendix B First-order logic system in SPASS format of $W_{SA} - T_Q(W_{SA})$

```
begin_problem(class).
  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
expression (forall([x],or(SoftwareArchitecture(x),or(Component(x),or(Connection(x),or(Interface(x),RefComponent(x)))))).
%==============(A.2) Disjointness of classification
expression(forany([x],contains(SoftwareArchitecture(x),not(Component(x))))).
expression(forany([x],contains(SoftwareArchitecture(x),not(Connection(x))))).
expression (forany([x],contains(SoftwareArchitecture(x),not(Interface(x))))).
expression (forany([x],contains(SoftwareArchitecture(x),not(RefComponent(x))))).
expression (forany([x],contains(Component(x),not(Connection(x))))).
expression (forany([x],contains(Component(x),not(Interface(x))))).
expression(forany([x],contains(Component(x),not(RefComponent(x))))).
expression(forany([x],contains(Connection(x),not(Interface(x))))).
expression(forany([x],contains(Connection(x),not(RefComponent(x))))).
expression(forany([x],contains(Interface(x),not(RefComponent(x))))).
%==============(A.3) Enum Attribute Type Constraint
expression(forany([x,y],contains(and (ComType(x,y),Component(x)),or(equal(y,commerce),or(equal(y,open-source),equal(y,self-
develop)))))))).
%==============(A.4) Not Enum Attribute Type Constraint
expression(forany([x,y],contains(and (ComPrice(x,y),Component(x)),equal(y,DblTrue)))).
%==============(A.5) Not Association Type Constraint
expression(forany([x,y],contains(AttachInfToCom(x,y),and(Interface(x),Component(y))))).
expression(forany([x,y,z],contains(and( AttachInfToCom(x,y),AttachInfToCom(x,z)),equal(y,z)))).
expression(forany([x,y],contains(AttachInfToCon(x,y),and(Interface(x),Connection(y))))).
expression(forany([x,y,z],contains(and( AttachInfToCon(x,y),AttachInfToCon(x,z)),equal(y,z)))).
expression(forany([x,y],contains(ComRef(x,y),and (RefComponent(x),Interface(y))))).
expression(forany([x,y,z],contains(and ( ComRef(x,y),ComRef(x,z)),equal(y,z)))).
%==============(A.6) Association Type Constraint
expression(forany([x,y],contains(InfAssociation(x,y),and (Interface(x),Interface(y))))).
%==============(A.7.-A.8) Multiplicity Constraint
expression(forany([x,y,z],contains(and(InfAssociation(y,x),InfAssociation(z,x)),equal(y,z)))).
expression(forany([x,y,z],contains(and(InfAssociation(x,y),InfAssociation(x,z)),equal(y,z)))).
%==============(A.9.) EnumNotEqual Constraint
expression(not(equal(commerce,open-source))).
expression(not(equal(commerce,self-develop))).
expression(not(equal(open-source,self-develop))).
%==============(A.10.) NotEnumNotEqual Constraint
expression(not(equal(DblTrue ,DblFalse))).
end_of_list.
end_problem.
```

REFERENCES

1. Object Management Group, MDA guide version 2.0, 2014, http://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf.
2. Steven K, Juha-Pekka T, Domain-specific modeling: Enabling full code generation, New Jersey, USA: John Wiley & Sons, 2008.
3. Jackson.E.K and Sztipanovits.J, Formalizing the Structural Semantics of Domain-Specific Modeling Languages, Journal of Software and Systems Modeling, 2008, 8(4): 451-478.
4. Chomsky N, On certain formal properties of grammars, Information and Control 2, 2 (1959), pp. 137–167.
5. VARR'O, D., AND PATARICZA, A. Vpm: A visual, precise and multilevel metamodeling framework for describing mathematical domains and uml (the mathematics of metamodeling is metamodeling mathematics). Software and System Modeling 2, 3 (2003), 187–210.
6. Vanderbilt University, GME User's Manual, 2016, http://www.isis.vanderbilt.edu/Projects/gme/.
7. Object Management Group, Unified Modeling Language: version 2.5, 2015, http://www.omg.org/spec/UML/2.5/PDF.
8. Object Management Group, Meta Object Facility Specification version 2.5, 2015, http://www.omg.org/spec/MOF/2.5/PDF.
9. Object Management Group, Object Constraint Language Version 2.4, 2014, http://www.omg.org/spec/OCL/2.4/PDF.
10. Sun XP, A Research of Visual Domain-Specific Meta-Modeling Language and Its Instantiation, Ph.D. Thesis, Yunnan University, Kunming, 2010.

11. Shan L, Zhu H, Semantics of Metamodels in UML, in Proceedings of the Sixth International Symposium on Theoretical Aspects of Software Engineering, Tianjin, China, IEEE-CPS, pp. 55-62, 2009.
12. Mustafa Al-Lail, et al, An Approach to Analyzing Temporal Properties in UML Class Models, in Proceedings of MoDeVVa 2013, Miami, Florida, USA, ACM, pp. 77–86, 2013.
13. Ruzhen Dong, et al., rCOS: Defining Meanings of Component-Based Software Architectures, in Proceedings of International Training School on Software Engineering Held at ICTAC 2013, Shanghai, China, Springer, pp. 1-66, 2013.
14. Faiez Zalila, et al., Formal Verification Integration Approach for DSML, in Proceedings of International Conference on Model Driven Engineering Languages and Systems, Miami, Florida, USA, ACM, pp. 336-351, 2013.
15. Ethan K. Jackson and Wolfram Schulte, "Understanding Specification Languages through Their Model Theory", in Proceedings of 17th Monterey Workshop, Oxford, UK, ACM/IEEE, pp. 396-415, 2012.
16. Ethan K. Jackson and Wolfram Schulte, FORMULA 2.0: A Language for Formal Specifications, in Proceedings of International Training School on Software Engineering Held at ICTAC 2013, Shanghai, China, Springer, pp. 156-206, 2013.
17. Tao Jiang, et al., A formal representation of the structural semantics of Domain-Specific Modeling Language, in Proceedings of the IEEE International Conference on Computer Science & Automation Engineering (CSAE 2012), Zhangjiajie, China, IEEE, volume 3, pp. 747-751, 2012.
18. Cheng MZ and Yu JW, Logic foundation—first-order logic and first-order theory, Chinese People University Press, Beijing, 2003.
19. Tao Jiang, an Approach for Automatically Reasoning Consistency of Domain-Specific Modelling Language, Lecture notes in computer science, 8818: p. 295-306, 2014.
20. Max-Planck-Institut Informatik, SPASS Tutorial, 2015, http://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover/tutorial/.
21. Tao Jiang, An Approach for Automatically Verifying Metamodels Consistency, International Journal of Simulation Systems, Science & Technology, 17(27): pp. 20.1-20.7, 2016.
22. W.McCune, Prover9 Manual and Examples, 2015, http://www.cs.unm.edu/~mccune/prover9/.

Biography

Tao Jiang 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. 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.