Model Mapping Approach Based on Ontology Semantics

Jinkui Hou and Yuyan Zhang
School of Computer Engineering, Weifang University, Weifang, China
Email: jkhou@163.com, zyytt@yeah.net

Abstract—The mapping relations between different models are the foundation for model transformation in model-driven software development. On the basis of ontology semantics, model mappings between different levels are classified by using structural semantics of modeling languages. The general definition process for mapping relations is explored, and the principles of structure mapping are proposed subsequently. The approach is further illustrated by the mapping relations from class model of object oriented modeling language to the C programming codes. The application research shows that the approach provides a theoretical guidance for the realization of model mapping, and thus can make an effective support to model-driven software development.

Index Terms—Model-Driven Development, Model Mapping, Formal Semantics, Ontology Semantics; Structural Semantics

I. INTRODUCTION

Currently, Model-Driven Software Development (MDSD) is highly regarded as a development paradigm among developers and researchers [1, 2]. Numerous research institutions and enterprises have been investing a large amount of money and manpower in the MDSD study. Currently, a number of products based on MDSD have proved that a lot of benefits can be obtained from it, such as rapid development, architecture advantages, improvement of codes consistency and maintainability, enhancement of system’s portability across different middleware vendors, and it also shows great potential in these areas [3].

Generally, model-driven development is a technique for addressing complex development challenges by dealing with complexity through abstractions provided by models. Modeling and model transformation constitute the core of model-driven development. Models can be refined and finally be transformed into a technical implementation, i.e., a software system. By using this technique, complex systems are modeled at different levels of specificity. As the development program proceeds, the model undergoes a series of transformations, with each transformation adding levels of specificity and detail [1]. For the development of complex systems, MDSD begins with the black-box specification of the system and, through a rigorous process of transformation, creates a model of the system. In the ultimate result, the model is realized with tested and functioning system components.

On the whole, the existing and proposed approaches for model transformation can be classified into five categories [4]: Template-based approaches, Target-structure-driven approaches, Graph-transformation-based approaches, Relational approaches and Transformation implemented using XSLT. Most of these approaches focus on providing a concrete solution for the transformation between models at different abstract levels, and there is currently no mature foundation for the definition of mapping rules as well as cardinal principles to validate the mapping relations between such models. More in-depth study and formal methods about this issue are expected to support complex model transformations.

Semantic structure is the semantic elements expressed by the model and the relationship between these semantic elements. There are many differences between different modeling languages, such as the concerns of analysis, the design methods and the expression ways for semantic features [5], and which have a profound impact on the organization structure and information expression of models at different abstract levels. These differences make the mapping relations between models very complex and difficult to define directly. In this paper, on the basis of ontology semantics, model mappings between different levels are classified by using structural semantics of modeling languages. The general definition process for mapping relations is explored, and accordingly the principles of structure mapping are proposed.

This paper is organized as follows: in the next section, the semantic features of modeling language and model mapping are discussed briefly; a classification for different level mappings, the general definition process and the cardinal principles for mapping relations are given in Section III; A case study is shown in Section IV to further illustrated the ideas; The paper ends with conclusions and future works.

II. MODELING LANGUAGE AND MAPPING RELATIONS

According to [5] and [6], a modeling language is composed of three parts: a precisely defined syntax, an unambiguously defined semantics and semantic mapping from the syntactic elements to the semantic domain. The syntax includes some syntactic notations based on its semantic types and some operation rules for these
notations. Semantics defines the meaning of a notation: what information do the expressions in the notation describe. Semantic domain is a well-understood domain of elements that describe the important properties of what we are trying to define using a language. Semantic mapping is a mapping that relates each syntactic construct to a construct of the semantic domain, and it can be used to explain new constructs in terms of known constructs. In the context of model-driven software development, the semantic domain involves many idiosyncrasies of software and hardware systems and components of such systems. These idiosyncrasies are based on concept structure and functional features of computer system, such as the process idiosyncrasy, interruption idiosyncrasy, data storage characteristics, control features, message features, events characteristics, thread idiosyncrasy, transaction idiosyncrasy, and so on. These idiosyncrasies and their composition constitute the common semantic foundation for modeling languages at different levels.

A. Semantics of Modeling Language

The difference between different modeling descriptions are represented at three levels: syntax, semantics and structure. The syntactic gap refers to the difference among the date types and styles in different models. There also exist difference within the date structures, link ports and patterns of different models, and which is called isomeric features at structure levels. The semantic gap means that the meanings of the terminologies used in different domains are not the same. For instance, platform independent models described in UML may include many graphic units, and the target model based on certain platform generally has many particular application patterns. The distance between them is more significant. The gap between the modeling languages can be narrowed using formalism extensions, but cannot be completely eliminated [4]. The fundamental solution for the problem of the gap seems to be the creation of effective semantic mapping mechanism, so to ensure that the equivalent representations for the system can be obtained [7]. This mapping mechanism should be in a formalized way, which is convenient for humans’ understanding and computer’s processing.

A model is a representation of the function, structure and behavior of an application or system in a given formalism [4]. Model-driven development is a software engineering approach consisting of the application of models and model technologies to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying (making easier) and formalizing (standardizing, so that automation is possible) the various activities and tasks that comprise the software life cycle [1].

In principle, various modeling languages (or programming languages) is equivalent in some certain technical sense. Each language can be used to write anything that other languages could write, even though there may be some awkward solutions. If some very useful features are not provided directly in a language, we can generally get a simulation through a deliberate (not compulsory) way. For example, in the absence of abstract mechanisms for the language, the notes and the naming rules can be used to simulate the module structure. The iterator can be simulated by functions and static variables. Starting from the recursive pseudo-codes, an iterative program can be derived by manual transformation in the context short of recursive mechanism. If a language has no mechanisms for named constants and enumeration types, some variables once initialized and no longer modified are used for better readability and maintainability. Semantics is the meaning of information, which is relevant with its context.

Semantics concerns the meanings of modeling elements, such as signs, symbols, and the words that represent them. Semantic structure is the semantic elements expressed by the model and the relationship between these semantic elements. A definition on consistency of model mapping is given bellow according to [7], [8] and [9].

Definition 1 Model mapping consistency. LA and LB respectively represent modeling languages at different levels of abstraction. MA is the model defined with LA, and MB is the model defined with LB. The set of semantic features of MA is noted as $[M_A]$, and the set of semantic features of MB is noted as $[M_B]$. The mapping M:MA→MB is called strong consistency, if and only if $[M_A]=[M_B]$. For a semantic collection $\rho\subseteq[M_A]$, the mapping M:MA→MB is called weak consistency if and only if $\rho\subseteq[M_B]$. A consistency condition can be defined within the syntax expression based on a common semantic domain. In general, one distinguishes between two kinds of semantic consistency. Horizontal consistency problems exist for a set of models that describe the same aspect of a system from different points of view, potentially using different languages. It has to be ensured that these models do not contain contradictory concepts. Vertical consistency problems exist for models describing the same concept at different levels of abstraction. If a model is refined, it has to be guaranteed that the refined model does not contradict to the specifications of the more abstract model. An example for horizontal consistency is a UML state chart and a sequence diagram that describe the same process but from different angles, while vertical consistency may be illustrated with a state chart $S$ and its refined version $S'$, where it may be required that all states that are reachable in $S$ have to be reachable in $S'$ as well.

B. Mapping Relations

Ontology mapping is a method to solve the knowledge sharing and reuse between different ontology, in which the aim is to find out the semantic association between entities of different ontology, and to express it in formal way [7]. According to the definition of ontology, the types of ontology mapping are concept to concept, concept to property, property to property, context and constraint, etc.

In model-driven software development, a model is defined as a collection of all the artifacts that describe the system [10]. A transformation is the automatic generation of a target model from a source model, according to a
transformation definition [11]. From operational view, a transformation is a terminating algorithm that applies structural and/or semantic changes to a model or a set of models. From function view, a transformation is a function that maps a tuple of models from one or more domains onto another tuple of models in the same or different domains [1].

The mapping relations between models are not only the basis but also a guide to the transformation for it including the specification of the transformation from a model to another one. A mapping rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [12]. In MDSD-supported model transformation, there is a need for a clear understanding of the systems’ intention, concepts and symbols of different models as well as the relations between different conceptual constructs. The semantics of UML and some object-oriented programming languages should be identified firstly. Then, the static and dynamic mechanisms of the source and target modeling languages (or target programming languages) must be described using special describing mechanisms for semantics [13], and then semantic models for both the source and the target should be established respectively. Lastly, the corresponding relations between the modeling elements in the source language and target language may be built according to the semantic equivalence principle. Thereby, the mapping rules between semantic primitives of different modeling languages can be created in order to eventually realize the transformation from the source model to the target.

III. MODEL MAPPING

In MDSD, a model is a representation of the function, structure and behavior of an application or system in a given formalism [1]. Model transformations involve two or more models. The most typical example is a high-level abstraction model being transformed into a low-level abstracted and technology-dependent one. The target of this transformation can be models or programming codes, which is equivalent to the source. The "abstract level" refers to the degree of that the features of a modeling language separating from the technologies of any certain platform. Transformability between models means that the both ends can be translated into equivalent executable op-code without further possible errors.

A. Mapping Types

According to cooperative relations among components and ontology semantic mapping [14], the mapping relations between modes at the different abstract levels can be divided into two layers: data-type mapping and structure mapping. Data-types mapping refers to the definition of mapping rules that can be used for the transformation from date types defined in a modeling language to the types defined in another modeling language (or programming language), while the types at both ends are semantic equivalent. Structure mapping refers to the definition of mapping rules that can be used for the transformation from source modules to the modules built with target modeling languages (or programming languages), while the structure and semantic features as well as the functions of the both modules are equivalence or very similar.

In programming language, there are two kinds of data types from the point of construction view, which is respectively called atomic types and combination types [15]. The former is also called simple types, such as Integer, Characters, Boolean, Real, and so on. The latter is composed of two or more simple types, which is aggregated by some type constructors, such as record, array, set, etc. According to Definition 1, conform mapping occurs when the target modeling language has the corresponding date types, which semantic meaning is equivalent to the types in the source language. If there are no data types in the target language which is the direct counterpart for the source type, a simulated construction with equivalent meaning can be defined by using several kinds of basic types [16].

As first-class objects in structure descriptions, modeling elements can be transformed and refined [17]. The structure objects of the source model can be decomposed, aggregated, or composed in target model. From the view of structure granularity, there are three basic forms of mappings between models at different abstract levels: direct mapping, decomposition mapping and composition mapping, which are shown in Fig. 1.

As shown in Fig. 1 (a), the simplest structure mapping is one-to-one mapping. It maps a module to another module directly, such as the mapping from the concept “Class” in UML to the concept “class” in the Java programming language.

Definition 2. Direct mapping between structures is defined as a 4-tuple: \( M = (S, T, \rightarrow_{S} \rightarrow_{T}, d) \), where \( S \) and \( T \) respectively represent the source module and the target module. Herein, \( \rightarrow_{S} \) is the mapping relations from \( S \) to \( T \), while \( d \) is used to reflect the consistent degree of this mapping. The value of \( v \) mainly depends on the semantic similarity between the concepts of modules and their context, which may be varying in the range [0, 1]. The greater the value is, the higher consistent degree will be.

Structure mapping may be very complex for the gap between different modeling languages. The mappings across different granularity levels can be classified into decomposition mappings and composition mappings. A decomposition mapping is also called a “1-to-n” mapping, in which a source component at a given granularity level is mapped to a composite of several components at another smaller granularity level in the target model, i.e., the target \( T \) comprises several sub-structures denoted by \( \{T_{i}\}_{i \leq n} \), and which is shown in Fig. 1(b).

Definition 3. Decomposition mapping between structures is defined as a 5-tuple: \( M_{D} = (S, V, T, D_{T}, d) \), where

1. \( S \) represents the source module, and \( T \) represents the target module which is composed of some submodules noted as \( T_{i}(i \leq n) \);

2. \( V \) is a set of mapping expressions: \( V = \{ v_{i} \}_{i=1}^{n} \), where \( v_{i} \) represents the mapping from \( S \) to \( T_{i} \), which is a submodule of the target composition \( T \), and \( \rightarrow_{S} \rightarrow_{T} \) is a direct mapping;
(3) $D_T$ is a set in which the elements represent the dependency relations between the submodules of $T$: $D_T = \{ D_i \subseteq D_j \mid 1 \leq i, j \leq n, 1 \leq k \leq n(n-1)/2 \}$, where the number $i, j$ and $k$ respectively shows the order of the submodules $D_i$, $D_j$ and the dependency relation $\subseteq_k$ in the whole composition.

(4) $d$ is the consistent degree of the decomposition mapping.

As shown in Fig. 1 (c), a composition mapping is also called a “n-to-1” mapping, in which a composite of some descriptions at a given granularity level is mapped to a single target module. That is to say, the source of a composition mapping is a combination of some submodules (denoted by $\{ S_i \mid i \leq n \}$). In this case, we have the following definition.

Definition 4. Composition mapping between structures is defined as a 5-tuple: $M_c = (S, D, T, V, d)$, where

1. $S$ represents the source module which is composed of some submodules noted as $S_i$ ($1 \leq i \leq n$), and $T$ represents the target module;

2. $D$ is a set which elements represent the dependency relations between the submodules of $S$: $D = \{ S_i \subseteq S_j \mid 1 \leq i, j \leq n, 1 \leq k \leq n(n-1)/2 \}$, where the number $i, j$ and $k$ respectively shows the order of the submodules $S_i$, $S_j$ and the dependency relation $\subseteq_k$ in the whole composition;

3. $V$ is a set of mapping expressions: $V = \{ v_i \mid v_i : S_i \rightarrow T, 1 \leq i \leq n \}$, where $v_i$ represent the mapping from $S_i$ (which is a submodule of the source composition $S$) to the target $T$;

4. $d$ is the consistent degree of this composition mapping.

In practice, if the mapping relation between different modules belongs to “many-to-many”, an intermediate structure can be added to simplify the definition. That is to say, the “many-to-many” relation may be split up into a “many-to-one” relation (composition mapping) and a “one-to-many” relation (decomposition mapping).

B. Structure Integration

In the mapping from source model to target model, besides data-type mapping and structures mapping, the semantic features and constraint relations in the source are required to be maintained in the target [18].

Definition 5 Structure integration. Given a set $S = \{ S_i \mid 1 \leq i \leq n \}$, consists of $n$ different sub-modules, and $C = \{ C_i \mid 1 \leq i \leq n \}$, represents a finite set of constraint relations, we call module $A$ an integration of $S_i$ $(1 \leq i \leq n)$ if the following conditions are satisfied:

1. there exist $n$ mappings $M_i$ from $S_i$ to $S$, where $1 \leq i \leq n$;
2. $(\forall x, y \in S_i) \ x \preceq y \Rightarrow M_i(x) \preceq M_i(y)$, where $1 \leq i \leq n$, and the symbol $\preceq$ represents a certain kind of semantic hierarchical relations;
3. $(\forall x, y) \ (x \in S_i \setminus y \in S_j) \land (x \ op \ y) \in C \Rightarrow (M_i(x) \ op \ M_j(y)) \in C$, where $1 \leq i, j \leq n$, and here $op$ represents a certain kind of constraint relations.

The conditions given above mean that, the semantic hierarchical relations [19] in each sub-module should be still maintained in the integration. The last condition give the formal description for that the constraint relations between different sub-modules are also preserved in the integration module [20].

The mapping between different level models will not only carry out the data-type mapping and structure mapping from the source to the target. Meanwhile, in the target model, the various sub-components must keep semantic features and constraint relationship in the source model. Therefore, the semantic characteristics of each module and the constraint relationship between these modules should be integrated into an overall target structural model. Thereby, all mapping relations from the source model to the target model are obtained.

The mapping process between different models can be divided into the following three steps according to the definitions given above. The first step is called integration of direct mappings, and the primary work is to build an integrated link list in terms of the existing direct mappings (including data-type mapping). Herein, the direct mapping relations in complex mappings are also involved. The second step is called integration of complex mappings, and the main work is to build complex mapping relations between those modules with no direct mappings. Decomposition mappings and compositions mappings are all included. The third step is called systemic integration. The whole mapping relations between the source model and the target model will be achieved finally according to the semantic features and constraint relations among modules built within the former two steps.
IV. A CASE STUDY

The mapping relations from UML-based class model to C programming language model are used in the case study to illustrate the idea of defining model mappings which preserving related properties.

A. Features of Class Model

Class is the most important building block in object-oriented system, which is a mechanism used to describe the structure and behavior characteristics. It is a description of a group objects with same attributes, operations, relationships, and semantics. UML-based class model [21] represents a partial abstraction of a software system, in which model elements used to express semantic features mainly include class, attribute, operation, inheritance, aggregation, association, etc. Inheritance describes the relation of subclass-supersclass, which is used to build a hierarchical structure for modeling concepts. The cardinality of association relations is used to reflect the multiplicity while a kind of objects is taken as attributes in another class. Complex model can be constructed by using these basic features and their composition.

A pattern is a combination of a set of conceptual variables and the relevant constraints which modeling elements bound to the pattern must satisfy. Patterns mapping is a depiction for semantic equivalent relations between the elements within different patterns. The contents of the class-patterns mapping mainly include classes-mapping, attributes-mapping, operations-mapping, the mapping between class and attribute, the mapping between class and operation as well as the combinations of these mappings, which are shown briefly in Fig. 2.

In Fig.2, the symbol $\varnothing$ represents an empty set, and the arrows from each modeling element to it indicate the deletion of the elements, while the arrows from it to each modeling elements means the creation of the elements. Herein, only the basic mappings between class-patterns are taken into account, and the case will be more complex if the cardinality of associations is considered.

![Figure 2. The primitive mappings between class models](image)

B. Formal Semantics of Class Model

Formal semantics of class models and the related properties can make a help to the description of complex mappings between models at different levels [22]. Class pattern can be defined as a 4-tuple: $P_1=\langle E_C, C_i, C_r, F \rangle$, where $E_C$ is a set of modeling elements such as classes, attributes and operations, and $L_A$ represents the set composed of labels for associations while $L_I$ represents the set of labels for generalization or specialization. On this basis, we have the following expressions:

1. $A \subseteq E_C \times C_i \times E_C$ represents the set of association relations;
2. $R \subseteq E_C \times C_i \times E_C$ represents the set of generalization or specialization relations;
3. $F: A \rightarrow \{[0: 1], [1: 1], [1: N], [N: 1], [N: M]\}$ represents the set of functions which attach a cardinality to an association mentioned above.

Next, several properties of class model is discussed. The lowercase letters such as $a$, $b$, $c$, ... is used to represent the edges for attributes and operations, and the lowercase letters such as $p$, $q$, $r$, ... indicate classes and types. The edges standing for inheritance was represented by the capital letters such as $U$, $V$, etc. We also have the following expressions representing semantic constraints.

1. Given $A \subseteq E_C \times C_i \times E_C$, the expression $(p, a, q) \in A$ is noted as $p \rightarrow a q$, which means that all instances of class $p$ have an attribute $a$ whose type is $q$;
2. Given $R \subseteq E_C \times C_i \times E_C$, the expression $(p, U, q) \in R$ is noted as $p \Rightarrow q$, which means that $p$ is a specialization of $q$, i.e., all instances of class $p$ are instances of class $q$ at the same time;
3. For short, the expression $p \Rightarrow r_1 \Rightarrow r_2 \cdots \Rightarrow r_m \Rightarrow q$ or $p=q$ is noted as $p \Rightarrow q$.

Based on the expressions given above, several basic properties of class model can be derived as follows.

1. if $p \Rightarrow q_1$ and $p \Rightarrow q_2$, then $q_1=q_2$;
2. if $p \Rightarrow q$ and $q \Rightarrow r$, then $p \Rightarrow r$;
3. if $p \Rightarrow s$ and $s \Rightarrow r$, then $p \Rightarrow r$.

Property (1) shows that not only the related classes but also the names of attributes are used to depict a class. Property (2) shows the inherited characteristic of attributes in specialization relations, that is to say, if the class $p$ is a specialization of class $q$ and $a$ is an attribute of $q$, then $a$ is also an attribute of $p$. Property (3) shows the preserving characteristic of specialization relations in attributes from another perspective, i.e., if $p$ has an attribute $a$ whose type is $s$ and $s$ is a specialization of $r$, then the same can be said as $p$ has an attribute $a$ whose type is $r$.

C. Mapping from Class Model to C Structure Model

In object-oriented unify modeling language, Class is a composition of specific operations on specific data, which containing two categories: data and operations. In the C programming language, there are no direct data-types or structures as a match for Class, but we can get a simulation by using structs, prototypes, pointers and function pointers. Several mechanisms or features of object-oriented programming language can be simulated as follows.

1. The simulation for class can be defined using struct and function pointers, such as:

   ```c
   struct CLASS(R)
   double l;
   double w;
   ```
The key word CLASS is a predefined macros with parameters for the concept struct in C programming language. Its definition is shown as follows.

```
#define CLASS(type) 

typed struct type type; 

(2) Interface is a collection of operations, which specifies services provided by an object. The difference between the definition of interface and class is that interface contains only member functions, no data members, so Struct in C language also can be used to achieve interface mechanism.

(3) The simulation for function overloading can be achieved by sub-function calls in terms of the flag. The following is an example.

```
long overldFnc (int f, Prm *p){
    switch (f){
        case fg1: subFunc1(…);
        case fg2: subFunc2(…);
        …
    }
```

(4) Dependency relationship in UML describes the associations and constraints among model elements. In C language, we can implements the dependency relationship between two classes using one class as the parameter of a operation function of another function. The implementation code is shown as follows.

```
CLASS(B){
    void (*setB)(type *t);
}
```

(5) Encapsulation mechanism is used to protect the internal details of objects. In UML, there are several visibility options for the adaptation to different situations, such as private, package, public, etc. In the C programming language, this mechanism can be simulated as follows.

```
struct LB{
    int bg;
    char pv[1];
}
```

(6) Generalization relationship describes the relationship between an ordinary thing and the special class of this thing. It can be regarded as a kind of relationship between parent class and child class. For example, horse is one kind of animal, which has the general characters of animal, and the specific characters of itself. In C language, because the distribution of the elements of a struct in the memory has the same sequence as its statement, generalization relationship can be achieved by using the concept struct. What we need to do is to define the type of one property value as parent class in the child class. The implementation code is shown as follows.

```
CLASS(Animal){
    …
}
CLASS(Horse){
    Animal:animal;
    …
}
```

(7) Connection relationship describes the connection between one class and another class. It makes the visible property and means of one class to be used by another class, such as company and its employees. One company corresponds to some specific employees, and one employee corresponds to a specific company.

In the struct definition of C programming language for the simulation of class A, the associations from class A to class B can be simulated by adding a member pointer to a variable whose type is the struct simulating class B. If the cardinality is greater than 1, array of pointers or link list can be used in the description. The implementation code is shown as follows.

```
CLASS(Company) {
    Employee *employee[];
    …
}
```

(8) There are two specific connection relationships in UML diagram. Aggregation relationship is a kind of relationship between the whole and the part. The part can exist alone, leaving the whole. That is to say, the existence of part objects does not directly depend on the whole object. For example, a flock of horse is formed by numbers of horses. So we only need to define the pointer variable of the part object in the whole structure. The implementation code is shown as follows.

```
CLASS(HorseGroup){
    Horse *horse;
    void *setHorse(void *,Horse *){
        …
    }
```

In the relationship of UML diagram, composition relationship is more rigorous than aggregation relationship. Composition relationship is also called strong aggregation relationship, which shows the relationship between the whole and the part too. But in such case, the whole and the part is inseparable. The end-life of the whole also means the end-life of the part, just like the relationship between geese and its wings.

Because the whole and the part have the same lifetime in the connection relationship, the formation of the part
objects is implemented by the whole objects. Thereby, we only need to define a member whose property type is part in the Struct. By this means, we can get the part objects when we get the whole objects. The implementation code is shown as follows.

```
CLASS(Horse)
    ......
    Leg legs;
    ......
```

(9) The polymorphism mechanism can be simulated by using function pointers in the C programming language. The function’s prototype, which including descriptions for the parameters and the return value as well as the function’s meaning, are given by function pointers. Different functions may have the same name. Thereby, the pointer can point to different functions according to the requirements, and the mechanism for polymorphism is achieved.

(10) In the struct definition for the simulation of class A, the associations from class A to class B can be simulated by adding a member pointer to a variable whose type is the struct simulating class B, if the cardinality is greater than 1, array of pointers or link-list can be used in the description.

Many other features of object-oriented programming languages also can be simulated by the C programming language, such as delegates, forwarding messages, dynamic type checking, etc, and which are not listed for the limited space. The mapping relations from UML-based class model to the C programming language model are given briefly in TABLE 1.

<table>
<thead>
<tr>
<th>UML-based Class Model</th>
<th>The C Programming Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>class name</td>
<td>struct name</td>
</tr>
<tr>
<td>data members</td>
<td>struct members</td>
</tr>
<tr>
<td>member functions</td>
<td>functions</td>
</tr>
<tr>
<td>constructing functions</td>
<td>functions with a pointer parameter to the</td>
</tr>
<tr>
<td>virtual functions</td>
<td>function pointers</td>
</tr>
<tr>
<td>function overloading</td>
<td>sub-functions calls in terms of flag</td>
</tr>
<tr>
<td>associations</td>
<td>a member pointer to other struct variables</td>
</tr>
</tbody>
</table>

An actual mapping from a small portion of a class model to C programming language model is shown in Fig. 3. There are many ways for the realization of the UML-based model with structure programming language, and here is an easier one. The left side depicts a small portion of a class model, and the right side shows the counterparts in the C programming language. The real line represents the mapping relations. The dashed line reflects the constraint relations of the target model. As can be seen, directing mappings, decomposition mappings and composition mappings are all included.

V. CONCLUSION AND FUTURE WORK

In this paper, on the basis of ontology semantic mapping, a classification for different level mappings is proposed with the abstract analysis of modeling languages. A further study about the general definition process for mapping relations is explored, and the principles of structure mapping are studied subsequently. It can not only be a theoretical guidance to model transformation, but also can be a measurement for validating the mapping rules between models at different abstract levels, and thus to provide an effective support to model driven software development.

As far as future work is concerned, there are several directions that we would like to explore: (1) further to formalize the definitions of data-type mapping and structure mapping; (2) to study more about the semantic properties which should be preserved in model transformation; (3) to make a summary of the generic proving processes and propose the algorithms to strictly prove whether a transformation satisfies a property preservation constraint or not.

![Figure 3. Mapping from class model to C code model](image)

ACKNOWLEDGMENT

The authors are most grateful to the anonymous referees for their constructive and helpful comments on the earlier version of the manuscript that helped to improve the presentation of the paper considerably. This research is supported by the Scientific Research Foundation for the Excellent Middle-Aged and Youth Scientists of Shandong Province of China under Grant No. BS2011DX039, and the Foundation of Science-technology Development Project of Shandong Province of China under Grant No. 2011YD01042 and No. 2011YD01043.

REFERENCES


