Abstract—As semantic consistency becomes the most important criterion of the correctness of model transformations in model-driven software development, the definition, description, and proof of semantic property preservation become crucial. This paper extends typed category theory to provide a formal description approach for architecture model transformation. Category diagrams are used to provide formal semantics of component specifications and architecture models. The transformation between different levels of models is formally described by morphisms. The approach can be used for the description, analysis and judgment of property preservation to strengthen the understandability and traceability of model transformations. The application research shows that the approach captures the essence, process and requirements of model transformation, and thus can make an effective support for model-driven software development.

Index Terms—formal approach, model transformation, software architecture, component model

I. INTRODUCTION

The correctness of model transformations is a key issue of model-driven software development [1,2]. Beydeda et al. have given the general criteria about the correctness of model transformation, which are syntactic correctness, syntactic completeness, termination, confluence and semantic consistency [3]. Herein, syntactic completeness is not a necessary condition. There already have been comparatively mature solutions for the judgment of these criteria with the exception of semantic consistency. The judgment about syntactic correctness can be achieved by using planner algorithms. Termination criteria can be analyzed by designing the transform systems as layered-rule-systems, and confluence criteria can be analyzed by using critical pair analysis for graph-based transformation systems.

However, there are currently no mature theoretical foundations and verification tools on the analysis and judgment about semantic consistency of model transformations.

All the facts show that the lacking of description and calculation approaches for semantic properties currently is the main lacking theory of model driven software development, and to build a theory for semantic description and calculation becomes the basis and urgency for its healthy and rapid development [4-6]. Consequently, a unified framework for the semantic description of the models and their mappings seems imperative. In our preliminary work [7], algebraic specification has been used to describe component models. The description framework is further refined and improved in this paper to provide precise semantics for architecture models and their mapping relations. On this basis, the semantic properties that should be preserved through model transformations are analyzed and discussed.

The rest of this paper is organized as follows. In Section II, formal descriptions of component models are presented on the basis of typed category theory. A discussion about the mapping relations between component models at different abstract levels are developed in Section III. The semantic properties that should be preserved through model transformations are analyzed and discussed in Section IV. A case study about a simple communication system is shown in Section V to further interpret the ideas. The paper ends with conclusions and future works.

II. FORMAL DESCRIPTION OF COMPONENT MODEL

In architecture-centric model-driven development, models are descriptions for component-based specifications, which are required to support transformation and composition while preserving semantic properties [8]. Architecture models and their mapping relations between different levels must be described precisely, so that the property preservation can be proved rigorously. In this paper, category theory is used as a basic framework for the formal description of component models. The basic knowledge about category
theory can be found in [9], which is not repeated in this paper.

A. Component Model

In the description and verification of a system’s semantic features, connectors are generally considered as same entities as components, and their definitions also require interaction ports and pertinent behavior protocols. Therefore, from the point of describing and verifying semantic properties, the distinction between component and connector is often subtle [10]. In order to maintain regularity and simplicity, we do not distinguish between these categories at the specification level, and both component and connector are generically called component.

To support more abstract levels of architecture description, we use a specification \(<\Sigma, \Phi>\) to represent the data types available in a certain architecture description language in a given environment, where \(\Sigma = S\times \Theta\) indicates the signature in the usual algebraic sense, i.e., \(S\) is a set of sort symbols and \(\Theta\) is an \(S\times S\)-indexed family of function symbols, together with a set \(\Phi\) of (first-order) axioms over \((S, \Theta)\) defining the properties of the operations. Particularly, at the level of programming language, \(<\Sigma, \Phi>\) can be taken as an abstraction of the properties of the data types supported by that language.

**Definition 1 Port signature.** A component port signature is a pair \(PS = (\Gamma, \iota_0)\), where

1. \(\Gamma\subseteq M\times O\) is set of actions that may arise in the port description, where \(M\) represents a finite set of methods and \(O\) represents a finite set of results;

2. \(\iota_0: \Gamma\to \Sigma^\iota\) is a partial function, which gives the set of actions triggered by a given action.

In the following definitions, we use \(G(p)\) to represent the set of actions of a port \(p\).

**Definition 2 Component signature.** A component signature is a 6-tuple \(\theta = <\Sigma, A, \Gamma, \iota, \phi, D>\), where

1. \(\Sigma = S\times \Omega\) is a data signature in the usual algebraic sense;

2. \(A\) is an \(S\times S\)-indexed family of attribute symbols, in which each attribute is typed by a data sort in \(S\);

3. \(\Gamma\) is an \(S\)-indexed family of port symbols;

4. \(\phi: A\to \Sigma^\phi\) is a set of total functions, which shows the properties of component attributes, such as the data type, visibility, etc;

5. \(\phi: A\to \Sigma^\phi\) is a set of total functions, which shows the properties of component ports, such as the port type, the types of received messages, etc;

6. \(D: \Gamma\to \Sigma^\delta\) is a total function, and for each \(p\in \Gamma\), \(D(p)\) is the collection of attributes that can be affected via the port \(p\).

**Definition 3 Component specification.** A component specification \(CP\) is a pair \((\theta, \Delta)\), in which \(\theta\) is a component signature \(<\Sigma, A, \Gamma, \phi, \iota, \phi, D>\) and \(\Delta\), the body of the specification, is a quadruple \((I, F, B, \Phi)\), where

1. \(I\) is a set of \(\Sigma\)-propositions which constraining the initial values of the attributes;

2. \(F\) assigns to every port \(p\in \Gamma\) a non-deterministic command which relates all attributes in \(D(p)\) to the actions of \(G(p)\);

3. \(B\) assigns to every port \(p\in \Gamma\) a \(\Sigma\)-proposition as its guard, which represents the conditions and constraints that should be satisfied for achieving the objectives of the component;

4. \(\Phi\) is a finite set of \(\theta\)-formulae (the axioms of the description), which represents the functional and non-functional objectives of the component.

The functions of a component not only consist of input-output conversions, but also have an impact on its external world. The semantic interpretation of a component specification is given in terms of transition systems.

**Definition 4 Component model.** A component model is a 2-tuple \(CM = (CP, \zeta)\), where \(CP = (\theta, \Delta)\), \(\theta = <\Sigma, A, \Gamma, \phi, \iota, \phi, D>\), \(\Delta = (I, F, B, \Phi)\), and \(\zeta = (\zeta, \psi, \nu)\) is a \(\theta\)-interpretation structure of \(CP\), such that

1. \(\zeta\) is a transition system \((W, w_0, E, \rightarrow)\), in which \(W\) represents a non-empty set of states; \(w_0\in W\) is the initial state; \(E\) indicates a non-empty set of events; \(\rightarrow\) is \(W\times W\) represents an \(E\)-indexed set of functions \(\rightarrow\) on \(W\) (i.e. state transition performed by each event);

2. \(\psi\) is a \(S\)-indexed family of maps \(\psi: A\to (W\to S)\), which returns the value of an attribute expression in a given state; \(A\) is a \(\theta\)-based attribute expression; \(S\subseteq S\) is the type set of expressions; a state consists of the values of component attributes; state transitions arise by the interpretation of functions of component ports;

3. \(\nu\): \(\{g| g\in G(p), p\in \Gamma\} \rightarrow \Sigma^\delta\);

4. \(\zeta, w_0\) = \(I\);

5. for each \(g\in \Phi\), \(\zeta\) = \(q\);

6. for each \(p\in \Gamma\), \(g\in G(p)\), \(a\in D(p)\), \(e\in \nu(g)\), and \(w, w'\in W\), \(w\rightarrow w'\); always \(\psi(a)(w)\rightarrow \{f(g,a)\}(w)\); herein \(\{f\}(w)\) means the semantic interpretation of the term \(f\) at the state \(w\) of \(\zeta\);

7. for each \(w\in W, p\in \Gamma\), \(g\in G(p)\), if \(e\in \nu(g)\) and \(w\rightarrow w'\) for some \(w'\in W\), then \((\zeta, w) = B(g)\).

That is to say, a component model contains a \(\theta\)-interpretation structure for its signature that enforces the assignments of the specification, only permits actions to occur when their guards are true, and for which the initial state satisfies the initialization constraints.

**B. Morphism**

The relationships between component specifications are represented by morphisms of category theory. The morphism types imply the different semantics of component relations. On the whole, the component morphisms of architecture models are classified into two kinds: interaction morphisms and composition morphisms. Interaction morphisms are used to describe the various interacting and dependence relations of components, which determine how a component’s service is combined with the services provided by other components in the system. Composition morphisms depict a kind of structure-preserving mappings, which are used to establish the relationship that must exist between two component descriptions so that one of them may be considered as a sub-component of the other.

The mechanisms of component relationships are provided and determined by the component models or
their implementation platform. A certain component relation is restricted by the component types and relation mechanisms. Component interaction types are abstractions that encapsulate component communication, coordination, and mediation decision, which depict the interaction protocols.

Definition 5 Component signature interaction morphism. Given two component signatures \( \theta_1: \Sigma_1, A_1, \Gamma_1, f_1, \rho_1, D_1, \rho_2, \Gamma_2, f_2, \rho_2, D_2, \rho_3, \), an interaction morphism \( \sigma: \theta_1 \rightarrow \theta_2 \) from \( \theta_1 \) to \( \theta_2 \) consists of a pair \( (\sigma_{\rho_1}, \sigma_{\rho_2}) \), where

1. \( \sigma_{\rho_1}: A_1 \rightarrow A_2 \) is a partial function satisfying
   - \( \exists a \in A_1, \text{ sort}(a) = \text{sort}(\sigma_{\rho_1}(a)) \); here, \( \text{sort} \) indicates the function for data types, and \( =_r \) represents the compatibility relationship of types;
   - \( \exists a \in A_1, f_1(a) = f_2(\sigma_{\rho_1}(a)) \), \( _\rho_1 \) means the consistency relations between property descriptions;

2. \( \sigma_{\rho_2}: A_2 \rightarrow A_1 \) is a partial function satisfying
   - \( \exists q \in A_1, f_1(q) = f_2(\sigma_{\rho_2}(q)) \); (3)
   - \( \exists q \in A_1, \text{sort}(q) = \text{sort}(\sigma_{\rho_2}(q)) \);

The function \( \sigma_{\rho_1} \) specifies the corresponding relations between attributes of the two interactive components, and the partial mapping \( \sigma_{\rho_2} \) identifies for each port of the component the corresponding port of the other component. Since not all attributes (or ports) of a component are necessarily involved in its interactions, both mappings are partial. Interaction morphism implies a synchronization point in the actions of the two components, and actions of the interactive ports constitute synchronization sets of their actions.

Definition 6 Component specification interaction morphism. Given two component specifications \( CP_1 = \langle \theta_1, A_1 \rangle \) and \( CP_2 = \langle \theta_2, D_2, \rho_3, \rangle \), in which \( \theta_1: \Sigma_1, A_1, \Gamma_1, f_1, \rho_1, D_1, \rho_3 \), \( \theta_2: \Sigma_2, A_2, \Gamma_2, f_2, \rho_2, D_2, \rho_3 \), an interaction morphism \( \omega: CP_1 \rightarrow CP_2 \) from \( CP_1 \) to \( CP_2 \) is a signature interaction morphism \( \theta_1 \rightarrow \theta_2 \) such that

1. \( \exists q \in \Phi_1, \omega(q) \in \Phi_2 \); (4)
2. \( \exists q \in \Phi_1, a \in D_1(q), F_1(\sigma(p), \sigma(a)) = \omega(F_2(p, a)) \); (5)
3. \( \exists a \in A_2, \omega(a) \in A_1 \); (6)
4. \( \exists q \in \Phi_2, B_2(\sigma(p)) = \omega(B_1(q)) \).

The first condition given above guarantees the functional and non-functional objectives should be preserved. The second condition means that the effects of the instructions can only be preserved or made more deterministic, and the third condition indicates that some initialization conditions are preserved. The last condition allows relevant guards to be strengthened but not to be weakened.

Composition morphisms are used to support hierarchical system design.

Definition 7 Component signature composition morphism. Given two component signatures \( \theta_1: \Sigma_1, A_1, \Gamma_1, f_1, \rho_1, D_1, \rho_3 \), \( \theta_2: \Sigma_2, A_2, \Gamma_2, f_2, \rho_2, D_2, \rho_3 \), a composition morphism \( \sigma: \theta_1 \rightarrow \theta_2 \) from \( \theta_1 \) to \( \theta_2 \) consists of

1. An algebraic signature mapping \( \sigma_{\rho_1}: \Sigma_1 \rightarrow \Sigma_2 \), which is injective;
2. An attribute mapping \( \sigma_{\rho_1}: A_1 \rightarrow A_2 \), such that for each \( f, s_1, \ldots, s_n \rightarrow s \in A_1 \), there exists an attribute symbol \( \sigma_{\rho_1}(f): \sigma_1(s_1), \ldots, \sigma_1(s_n) \rightarrow \sigma_2(s) \) in \( A_2 \), i.e. \( \sigma_1(A_1) \subseteq A_2 \);
Definition 9 Coproduct of signatures. Given two component signatures \( \theta_1 = \Sigma_1, A_1, \Gamma_1, f_{a_1}, f_{p_1}, D_1 \rangle \) and \( \theta_2 = \Sigma_2, A_2, \Gamma_2, f_{a_2}, f_{p_2}, D_2 \rangle \), the coproduct of \( \theta_1 \) and \( \theta_2 \) is given by the signature \( \theta = \theta_1 \uplus \theta_2 = \Sigma, A, \Gamma, f_a, f_p, D \rangle \) and two composition morphisms \( \sigma: \theta_1 \to \theta \) and \( \sigma_2: \theta_2 \to \theta \), where

1. \( (\Sigma, \sigma_1, \sigma_2) \) is the disjoint union of \( \Sigma_1 \) and \( \Sigma_2 \), in which \( \sigma(a) = \sigma_1(a) \) if \( a \in \Sigma_1 \) and \( \sigma(a) = \sigma_2(a) \) if \( a \in \Sigma_2 \);
2. \( (A, \sigma_1, \sigma_2) \) is the disjoint union of \( A_1 \) and \( A_2 \), in which \( \sigma(a_1) = A_1 \to A \), \( \sigma(a_2) = A_2 \to A \);
3. \( (\Gamma, \sigma_1, \sigma_2) \) is the disjoint union of \( \Gamma_1 \) and \( \Gamma_2 \), in which \( \sigma(a_1) = \Gamma_1 \to \Gamma \), \( \sigma(a_2) = \Gamma_2 \to \Gamma \);
4. \( \forall a \in A_1, i=1,2, f(a(a)) = f_{a_1}(a) \);
5. \( \forall p \in \Gamma_1, i=1,2, f_p(a(p)) = f_{p_1}(p) \);
6. \( \forall p \in \Gamma_1, i=1,2, D(a(p)) = D_{\Gamma_i}(p) \).

The conditions given above guarantee that: (1) the coproduct of \( \theta_1 \) and \( \theta_2 \) constitutes the coproduct of \( \theta_1 \) and \( \theta_2 \);

(1.1) \( \Sigma \) is the amalgamated sum of \( \Sigma_1 \) and \( \Sigma_2 \);
(1.2) \( A \) is the amalgamated sum of \( A_1 \) and \( A_2 \);
(1.3) \( \Gamma \) is the amalgamated sum of \( \Gamma_1 \) and \( \Gamma_2 \);
(1.4) \( \forall a \in A_1, \sigma(a) = \sigma_1(a) \);
(1.5) \( \forall p \in \Gamma_1, \sigma(p) = \sigma_1(p) \);
(1.6) \( \forall p \in \Gamma_2, i=1,2, D(a(p)) = D_{\Gamma_i}(p) \);

(2.1) \( I = \omega_1(I_1) \cup \omega_2(I_2) \);
(2.2) \( \forall a \in D(p), \sigma(a) = \omega_1(F_1(a)) \);
(2.3) \( \forall a \in D(p), i=1,2, B(a(p)) = \omega_i(B_{\Gamma_i}(p)) \);
(2.4) \( \Phi = \Phi_1 \cup \Phi_2 \).

In most cases, the systems are put together by interconnecting components. Given two components \( C_1 \) and \( C_2 \), which interact with each other by sharing a public function modules \( C_3 \) (which also can be seen as a connector), then the pushout of category theory can be used to obtain the parallel composition of these components. A pushout of a pair of morphisms with the same source \( v_1: C_0 \to C_1 \) and \( v_2: C_0 \to C_2 \) in a category is an object \( C \) and a pair of morphisms \( u_1: C \to C_1 \) and \( u_2: C \to C_2 \), such that the square commutes, which is shown in Fig. 2: \( u^* v_1 = u^* v_2 \) and such that the following universal condition holds: for all objects \( C \) and all morphisms \( l_1: C_1 \to C \) and \( l_2: C_2 \to C \), there exists a unique morphism \( l: C \to C \) such that \( l = u^* l_1 \).

Definition 11 Pushout of specifications. Given three component specifications \( CP_1 = \theta_1, \theta_2, A_1 \rangle, \) \( CP_2 = \theta_2, \theta_2, A_2 \rangle, \) \( CP_3 = \theta_3, \theta_3, A_3 \rangle, \) where \( \theta_1 = \Sigma_1, A_1, \Gamma_1, f_{a_1}, f_{p_1}, D_1 \rangle \), \( \theta_2 = \Sigma_2, A_2, \Gamma_2, f_{a_2}, f_{p_2}, D_2 \rangle \), \( \theta_3 = \Sigma_3, A_3, \Gamma_3, f_{a_3}, f_{p_3}, D_3 \rangle \), \( \theta_3 \rangle \) is the disjoint union of \( \theta_3 \) and \( \theta_3 \), such that the square commutes, which is shown in Fig. 2: \( u^* v_1 = u^* v_2 \) and such that the following universal condition holds: for all objects \( C \) and all morphisms \( l_1: C_1 \to C \) and \( l_2: C_2 \to C \), there exists a unique morphism \( l: C \to C \) such that \( l = u^* l_1 \).
subcomponents and their relations. The diagram will be called a finitely co-complete category if its colimit exists [9]. Obviously, component specifications and their morphisms constitute a finitely co-complete category.

D. Architecture Model

This paper aims to formally describe architecture models and discuss the semantic consistency problem of model transformation using category theory. A suitable typed category for software architecture description therefore seems imperative. Typed category [11] is a kind of category in which both the objects and the morphisms can have types, and each type can be defined with a series of features.

Definition 12 Architecture model. An architecture model is a 5-tuple $AM = \langle CO, CR, CT, RT, RuleS \rangle$, where $CO$ is a collection of component instances as objects; $CR$ a collection of component-relationship instances as object morphisms defined over $CO$; $CT$ a collection of component specifications; $RT$ a collection of specification morphisms as relation-types defined over $CT$; $RuleS$ a set of rules for relation-type composition. In addition to the basic conditions of the definition of categories [9], the following conditions also have to be satisfied:

1. $CO = \{o| 1 \leq i \leq m, \text{sort}(o) \in CT\}$, where $\text{sort}$ represents a function which returns the type of an object;
2. $CR = \{r| 0 \leq j \leq m, \text{exists } a, b \in CO, r = (a, b, t), t = \text{sort}(r) \in RT\}$;
3. $RuleS: RT \times RT \rightarrow RT$, and for all $(t, s) \in \text{dom}(RuleS)$, the type $w = \text{sort}(s)$ is called the composed type of $t$ and $s$; and for all $r, s \in CR$, $r = (a, b, t), s = (c, d, s)$, there exists a composed morphism $r_\circ s = (a, c, w) \circ t_\circ s$;
4. For each $a \in CO$, there exists an identity morphism $r_a = (a, a, ss), w = \text{sort}(r_a) \in RT$; and for all $t, s \in RT$, always $w = \text{sort}(s) = w$;
5. For all $r_\circ (a, b, t), s = (b, c, s)$, $s_\circ (c, d, q)$, $s_\circ r_\circ (a, d, (bs)_q) = (a, d, (bs)_q s_\circ r_\circ (b, c, r) = (b, c, r) s_\circ r$;
6. For all $r_\circ (a, b, t) \in CR$, always $r_\circ s \circ r_\circ s = r_\circ s = r_\circ s$,

where $r_a = (a, a, ss), r_a = (a, a, ss)$.

In our sense of categorical diagram, an architecture model is a typed category composed of component specifications and their morphisms. The specification of the whole system is given by the colimit of the underlying diagrams. The semantics of the configuration diagram should be seen as an abstraction of the cooperative execution that is obtained by coordinating the local executions according to the interconnections, and hence, the systemic design can be regarded as a whole.

III. MODEL MAPPING

Model mapping in this paper especially indicates the mapping relations from the component specifications at a higher abstract level to the specifications at a lower one.

Definition 13 Component signature mapping. Given two component signatures $\theta_1=\Sigma_1, A_1, \Gamma_1, f_{\sigma_1}, f_{\beta_1}, D_{\Sigma_1}$ and $\theta_2=\Sigma_2, A_2, \Gamma_2, f_{\sigma_2}, f_{\beta_2}, D_{\Sigma_2}$, a mapping morphism $\sigma: \theta_1 \rightarrow \theta_2$, from $\theta_1$ to $\theta_2$ consists of

1. An algebraic signature mapping $\sigma_2: \Sigma_1 \rightarrow \Sigma_2$, which is injective;
2. A total function $\sigma_a: A_2 \rightarrow \text{Term}(A_2)$ which mapping the attributes of $\theta_2$ to the class of terms built from the attributes and the data type operations of $\theta_1$, such that
   (2.1) for each $a \in A_1$, sort$(a) = \sigma_2(\theta_1(a))$;
   (2.2) for each $a \in A_1, f_{\sigma_1}(a) = \sigma_2(f_{\beta_2}(\theta_1(a)))$;
   (2.3) $\sigma_m$ is injective;
3. A total function mapping $\sigma_\Sigma: \Gamma_1 \rightarrow \Gamma_2$ from the ports descriptions of $\theta_1$ to those ones of $\theta_2$, such that
   (3.1) for each $p \in \Gamma_1$, $f_{\sigma_1}(p) = \sigma_2(f_{\beta_2}(\theta_1(p)))$;
   (3.2) for each $p \in \Gamma_1$, $\sigma_2(D_2(p)) \subseteq D_2(\sigma_\Sigma(p))$;
   (3.3) $\sigma_\Sigma$ is injective.

The conditions given above show that the basic signature mapping morphism contains the mapping between data types, the mapping between component attributes and the mapping between component ports. A mapping morphism must ensure the consistency between attribute types, the consistency between port types and the consistency between the descriptions for their properties from one level of abstraction to another. The attributes affected by a certain port must be preserved through the morphism. It is noted that the mapping morphism does not change the border between the system and its environment and, hence, input ports can not be mapped to output ports. Moreover, a mapping morphism allows each attribute of the source model to be mapped to an attribute expression in the target model. Such an expression may involve some computations as captured through the use of operations from the underlying data types. Naturally, it requires that the sorts of attributes must be preserved consistently. The function $\sigma_a$ identifies for each port of $CP_1$, the corresponding port (or a port set) of $CP_2$. In the target models, some ports for which $\sigma_a$ is left undefined (the new ports introduced in the target) and some attributes which are not involved in $\sigma_a(A_1)$ (the new attributes introduced in the target) are generally included, which implies that more details about the platform and the realizations are introduced in target descriptions.

Based on the concepts of signature morphism and axiom transfer, the mapping morphisms between component specifications are defined as follows.

Definition 14 Component specification mapping morphism. Given two component specifications $CP_1 = \langle \theta_1, A_1 \rangle$ and $CP_2 = \langle \theta_2, A_2 \rangle$, in which $\theta_1 = \Sigma_1, A_1, \Gamma_1, f_{\sigma_1}, f_{\beta_1}, D_{\Sigma_1}$, $\theta_2 = \Sigma_2, A_2, \Gamma_2, f_{\sigma_2}, f_{\beta_2}, D_{\Sigma_2}$, $A_1 = \{I_1, F_1, B_1, \Phi_1\}$, $\theta_2 = \Sigma_2, A_2, \Gamma_2, f_{\sigma_2}, f_{\beta_2}, D_{\Sigma_2}$, $A_2 = \{I_2, F_2, B_2, \Phi_2\}$, a mapping morphism $\omega: CP_1 \rightarrow CP_2$ from $CP_1$ to $CP_2$ is a signature mapping morphism $\sigma: \theta_1 \rightarrow \theta_2$, such that

1. For all $q \in \Phi_1$, $\omega(q) \in \Phi_2$. That is to say, if $q \in \Phi_1$, then $\zeta_2 = q = \omega(q)$, in which $\zeta_1$ and $\zeta_2$ respectively are $\theta$-interpretation structures of $CP_1$ and $CP_2$;
2. For each $p \in \Gamma_1$, $\omega(D(p)) = \sigma_2(D(p)) = \omega(D(p), a)$;
3. $I_2 \subseteq \omega(I_1)$;
4. For each $p \in \Gamma_1$, $B_2(\sigma(p)) \subseteq \omega_B(\omega(p))$.

Mapping morphisms reflect the relations between high-level specifications and concrete realizations. The first condition guarantees that all the functional and non-functional objectives of $CP_1$ must be preserved in $CP_2$. The second condition means that the effects of the instructions can only be preserved or made more
deterministic, and the third condition indicates that the initialization conditions are preserved. The fourth condition allows guards to be strengthened but not to be weakened.

IV. PROPERTY PRESERVATION OF MODEL TRANSFORMATIONS

In order to be of practical value, model transformations of software development must have some prerequisites, of which the most important is that the semantic consistency between source models and target models must be maintained. In other words, target models must preserve the semantic properties of source models [12].

Generally speaking, there are three kinds of semantic transfer arising in an architecture model transformation: behavior transfer, structure transfer, axiom transfer [13-15]. Structure transfer means the change of source component models on the structural aspect, which includes the introduction of new fine-grained structures, port-adding, data type transformation and topological extension, etc. Behavioral semantics indicates the transition of system states, which can be divided into external behavior semantics and internal behavior semantics. As for component model transformations, behavior transfer requires that the external behaviors are preserved. Axiom transfer means the transformation of the constraints imposed on architecture models. Herein, the constraints can be structural constraints, behavioral restrictions, or other constraints of non-functional aspects. Architecture model transformation generally is a combination of all these aspects. For example, not only the change on the structural aspect, but also the transfer of behavior descriptions and the corresponding constraints will be arising while a single component is being transformed to a composite of some subcomponents [16,17].

In the typed category based architecture model, the structural semantics is represented within categorical diagrams which depicts the architecture configuration and specifies the components and their interconnections. In order to analyze the impact of a model transformation on the system’s organizational structure, we can first analyze the dependency relations of components according to their interconnections. The component’s service-providing ports may rely on its own service-requiring ports, and its service-requiring ports may depend on some service-providing ports of other components. This kind of relationship of dependency is transitive [18]. Thereby, all the components which are potentially affected by a given component can be obtained by calculating the transitive closure of component port dependencies, and consequently the dependency graphs of components can also be achieved. All indirect dependency relations can be translated into direct dependencies firstly due to that the morphisms of category theory meet the associative law, and then inspect each node pair of the both graphs.

The semantic properties of a component model $CP$ (denoted as $R(CP)$), consist of the logics conveyed by the attributes, the actions and the axioms of its specification.

**Definition 15 Semantic preservation.** Given two component specifications $CP_1$, $CP_2 \in CT$, $R(CP_i)=\{s_i| i=1,...,n\}$ denotes the set of all the semantic properties inferred from $CP_i$. As for the mapping morphism $m:CP_1 \rightarrow CP_2$ and a given semantic requirements $L \subseteq R(CP_2)$, if every semantic property $s \in L$ which can be inferred from $CP_1$ also can be inferred from $CP_2$, then the semantic properties of $CP_1$ are called preserved in $CP_2$ under the selection by $L$ through the mapping $m$, i.e., the mapping morphism $m$ selectively preserves semantic properties. The semantic properties of $CP_i$ are called fully preserved in $CP_2$ whenever $L=R(CP_i)$.

Models are abstractions for the solution of problems, which give all essential properties of the solution and ignore the constraints about concrete implementations.[19,20] Therefore, from the viewpoint of property constructing of concept descriptions, models are certainly less than or really contained in their implementations.

V. A CASE STUDY

In this section, a simple communication system is used as a case study to illustrate the application of the theory and approach proposed in this paper. There are two components respectively named $Sender$ and $Receiver$ in the source model description. The $Sender$ first gets a message from the user at the sending end (named as $UserS$) and assigns the message and order, then sends the message to the $Receiver$. The $Receiver$ will forward the message to the user at the receiving end (named as $UserR$) after receiving the message.

The categorical diagram of the source model is shown as the left part of Fig. 3, where the morphisms $c_1$ and $c_2$ are composition morphisms, and the morphism $f_1$ is an interaction morphism. The specifications for the $Sender$ and the $Receiver$ are shown respectively in Fig. 4 and Fig. 5. Their colimit specification can be computed according to the definitions of Section II, which is shown in Fig. 6.
In the target platform environment, we assume that, the communication channel will be never interrupted, however the message can possibly be lost in the channel or be at fault, but the message will be certainly received eventually after times repeated sending. The acknowledgement message sent by the Receiver will be neither lost nor at fault. Therefore, a special channel between the Sender and the Receiver is introduced to transmit the acknowledgement message.

The corresponding target architecture model represented within a categorical diagram is shown as the right part of Fig. 3. The mapping relations from the source to the target are drawn with dashed arrows, which satisfy the commutative law of the category diagram, such as \( t_c \circ m_S = m_C \circ c_1, t_f \circ m_S = m_C \circ f_1 \), and so on. These properties show that the transformation following these mappings preserves consistency of the dependency relations among the components. Due to the limited space, the component specifications for the T_Sender, the T_Receiver and the T_channel are all omitted in this paper, and only their colimit specification (T_System) computed according to Definition 10 and Definition 11 is shown in Fig. 7. The detailed mapping relations from the source model to the target model can be obtained by comparing the attribute names, the port names and the names of axiomatic descriptions of the source specification with those names of the target, which are omitted in this paper for the limited space.

VI. CONCLUSION AND FUTURE WORK

Model transformations need to follow certain constraints to preserve some properties of models. The description and proof of these constraints is a hot topic of current research. However, to generally discuss the problem about property preservation is a very complicated issue, in which the determination of the semantic domain is a very important task. In this paper, category theory and algebraic specification are combined together to provide a unified semantic description framework for component-based architecture models and the mapping relations between them. On this basis, the semantic properties that should be preserved through model transformations are analyzed and discussed. The mapping description can be accomplished through a series of small and local mappings of which the results can be combined to form a larger composite architecture, i.e., we describe the mapping from an abstract model to a concrete implementation in an incremental way. The mapping descriptions provide the traceability of architectural design decisions. Our approach enables verification of property preservation in the way of theorem proving, which overcomes the shortcomings of model checking.

Another advantage of using category theory as a mathematical framework to formalize architectures is that the questions themselves can be formalized and resolved in terms that are independent of any specific ADL. Category theory supports the diagrammatic representation of component models that visualizes the relationships...
between components and the structural features, which can be used to strengthen the understandability and traceability of model transformations. The semantic description framework commendably captures the essence, process and requirements of MDD, which can be used as a new theoretical guidance for the cognition, design and semantic calculation of model transformations and model-driven development. The work does not only provide measurements for validating the mapping rules between different models, but also provide a theoretical guidance for the realization of model transformations, and thus make an effective support for model-driven software development.

However, the judgment on semantic consistency of model transformations is quite complicated, and this paper is only a preliminary study in this regard, in which the definition and description for semantic property preservation are still to be worked out in detail. As far as future work is concerned, there are several directions that we would like to explore: (1) further to formalize the definitions of component specifications and architecture models, and thus to strengthen the abilities of semantic expressiveness and consistent verification between models; (2) to study more about the semantic properties which should be preserved in model transformations for the enhancement of accuracy; (3) to make a summary of the generic proving processes and propose algorithms to strictly prove whether a transformation satisfies a property preservation constraint or not. Furthermore, the proving processes should be automatically achieved by computer and eventually be integrated into a MDA-supported modeling tool, and thus can support the design and verification of transformation rules.

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 was 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