Composition of Business Components Based on Behavior for Software Reuse

Fanchao Meng, Dechen Zhan and Xiaofei Xu

School of Computer Science Technology, Harbin Institute of Technology
P.O.Box 315, No. 92 West Da Zhi Street,
Harbin,P.R.China,15001

Mengfanchao74@163.com

Abstract

The capability to easily find useful components has become increasingly importance in software reuse field. Current researches on component retrieval are mostly limited to retrieve the component whose function is most close to the user requirement. However, in the process of component retrieval, we frequently face the problem that the individual component can't completely meet user's requirement. This paper proposes a method of composition of business components based on behavior, our purpose is to settle the problem that individual component can't completely meet the user requirement. In this study, deterministic finite state machine is used for modeling the behavior specification of business component. The composition of business components can be represented as the product of deterministic finite state machines whose result can be regarded as a nondeterministic finite state machine. Therefore the problem of component retrieval can be transformed into the matching between nondeterministic finite state machine and deterministic finite state machine, and behavior mapping graph is used to check the existence of composition of business components. Through the composition of business components, we can extract the behaviors in accord with user requirement from a set of candidate business components, which increase the reuse degree and reduce the cost of software development.

Keyword: business component, composition, behavior specification matching, behavior mapping graph

I. Introduction

Component-Based Software Development (CBSD) is a key technology to tackling the rapid development and software reuse of enterprise information system. CBSD is different from traditional methodology of software development, it emphasis much on retrieving reusable components from components repository. In the process of component retrieval, we have to face this problem that the individual component can not completely meet user requirement. Aim at the problem, a preferred approach to reducing the development costs combines these components that are functionally close to the functionality specified by the user. If the behavior of composition of these components can meet the user requirement, then we can extract these behaviors in accord with the requirement from these candidate components and reuse directly them, which can increase the reuse degree of components. A key of the composition of components is checking whether the behavior of composition of components can meet the user requirement. However, current researches on component retrieval pay most of their attentions to retrieving the component whose function is close to the user requirement [9][11][12][13][14][15], and ignore the composition of components, i.e., checking whether the behaviors of composition of a set of candidate components meet the user requirement and how to extract the behaviors in accord with the user requirement from these components.

Beyond classical signature based interface (such as EJB or CORBA), component behaviors specify called sequences accepted by a component (as given in a provided interfaces) or call sequences required by the component (as specified in the required interfaces). The benefit of including behavior specification in software component interfaces is widely agreed on [1][10]. Different approaches for behavior specifications have been proposed, ranging from state machine [1][2][3], Petri-nets[4][5][6], predicates[7][8], to process algebras[9][10]. Because finite state machines based approaches for behavior specifications can support automatic checking of compatibility, interoperability and substitutability, they are widely used for modeling the distributed systems and telecommunication systems. Luca de Alfaro [1] uses interface automata to capture the temporal aspects of software component interfaces. This formalism supports automatic compatibility checks between interface models and the refinement relation between abstract and concrete version of the same component. Osamu Shigo [3] describes a design method that defines the interface of a component by protocol state machine, and then systematically constructs a behavioral state machine of the component using the interface protocol state machines. This approach can check compatibility between behavior and its port protocols.

In this paper, we propose a method of composition of business components based on behavior, our purpose is to settle the problem that individual component can't completely meet the user requirement. In this study, deterministic finite state machine that extends the interface automata [1] is used to model the behavior specification of business component. Compare with interface automata and protocol state machine [2], we define final states in the deterministic finite state machine, so block must be considered in checking the compatibility when business components are combined. In interface automata, alternating simulation relation is used to define the refinement relation between abstract and concrete version of the same component. Because interface automata don't define the final states, refinement relation can't describe the language containment that ensures that the behavior of composition of business components can meet the behavior specified by user. Redondo[12] directly uses language containment to define the matching relation between user requirement and individual component. This definition is appropriate to describe the matching relation between two deterministic finite state machines, however, it isn't appropriate to describe the matching relation between deterministic finite state machine and nondeterministic finite state machine. Because different business components maybe include same or similar operations that don't belong to the shared operations among them, the behavior specification of composition of business components can be seen as a nondeterministic finite state machine. Therefore the language containment isn't appropriate to describe the behavior specification matching relation between user requirement and composition of business components.

Aim at the deficiencies of refinement relation and language containment, behavior specification matching relation is proposed to formalize the matching between user requirement and composition of business components. The behavior mapping graph is used to check the existence of composition of business components. To demonstrate the correctness and validity of the proposed method, we give corresponding theoretical proof and practical case.

Rest of this paper is organized as follows. In section II, we discuss related works. In section III, we propose our approach to modeling business components and the composition of business components. In section IV, at first, we introduce the process of business component retrieval, then propose our approach to checking the existence of composition of business components and extracting the behaviors in accord with user requirements. In section V, an example is given to explain our approach. Section VI gives a brief summary on this paper.

II. Related work

Besides finite state machine approaches, there are many approaches to modeling behavior specifications of components, including predicate, Petri-nets and process algebras. Each of these approaches has its specific benefits and weakness.

Predicates based approaches for specifying protocols [7][8] can describe protocols of arbitrary complexity. Unfortunately, this universality makes checking compatibility and substitutability of composition of components incomputable.

In [4][5], Petri-nets based approaches are used for modeling component behavior specifications. Nabil[4] presents a new and optimistic approach to the definition of component protocols compatibility and provides a framework for modeling component protocols together with their composition. In [5], a formal model of component interaction is proposed by representing component behaviors by labeled Petri nets. Component compatibility is established by determining those components which, when connected, are free of deadlock. While efficient algorithms exist for some Petri-net models to check global properties (such as liveness, absence of deadlocks, etc.) other properties which are important for component system (like interoperability and substitutability) can't be checked in general.

Process algebras being developed for describing the dynamic behavior of system are another candidate for modeling behavior specifications of components [9][10]. While being more expressive than finite state machines, they on the other hand lead to behavior specifications that have an infinite state space and can not be analyzed at all. Furthermore, for a software designer without specific formal background, finite state machines seem to be easier to create, modify and to understand.

III. Business Component Model

A. *Business Component*

Traditionally, a component is defined as a self-contained piece of software with well-defined interface or set of interfaces. A larger-grained component called a business component focuses on a business concept as the software implement of an autonomous business concept or business process [16]. A business component is a self-contained software construct, and it can provide a well-defined and well-known run-time interface, which means that it can be easily combined with other components to provide useful functionality.

Business components can be identified from enterprise model that includes a business goal, business objects, business rules and business process [17]. Compare with business object and business process, they belong to concepts of problem domain, and business component belongs to concept of solution domain. A business component represents and implements a business object or business process. In this paper, we focus on the interfaces provided and required by business components, and deterministic finite state machine is used to model the behavior specification of business component.

Definition 1. A business components can be defined as $C=(n,PI,RI,BS)$, where, n is the name of business component. $PI=\{PI_1,PI_2,\ldots,PI_m\}$ is the set of provided interfaces, and each provided interface consists of a set of provided operations. $RI=\{RI_1,RI_2,\ldots,RI_n\}$is the set of required interfaces, and each required interface consists of a set of required operations. BS is the behavior specification of business component that can be represented as a deterministic finite state machine, denoted as: $BS=(S_C,\Sigma_C,T_C,s_{C0},S_{CF})$, where S_C is a set of finite states; $\Sigma_C=\Sigma_C^P\cup\Sigma_C^H\cup\Sigma_C^R$ be the set of actions that represents operation calls or return values of operations, where, Σ_C^P is the set of input actions, Σ_C^H is the set of internal actions, and Σ_C^R is the set of output actions; $T_C\subseteq S_C\times\Sigma_C\times S_C$ is the set of state transitions; $s_{C0}\in S_C$ is the initial state, each business component includes only one initial state; $S_{CF}\subseteq S_C$ is the set of final states, and each business component includes at least one final state.

We use symbol $s\xrightarrow{a}s'$ to represent a transition of C, where, s, $s'\in S_C$, $a\in\Sigma_C$, $(s,a,s')\in T_C$. If $a\in\Sigma_C^P$ (resp. $a\in\Sigma_C^H$, $a\in\Sigma_C^R$), then $s\xrightarrow{a}s'$ is called an input (resp. internal, output) transition. We denote by $T^P{}_C=\{s \xrightarrow{a} s'|s,s'\in S_C,a\in\Sigma_C{}^P\}$, $T^H{}_C=\{s \xrightarrow{a} s'|s,s'\in S_C,a\in\Sigma_C{}^H\}$, and $T^R{}_C=\{s \xrightarrow{a} s'|$ $s,s'\in S_C$, $a\in\Sigma_C{}^R\}$ the sets of input, internal and output transitions.

An action $a\in\Sigma_C$ is enabled at state $s\in S_C$ if there is a transition $s\xrightarrow{a}s'\in T_C$ for some $s'\in S_C$. We indicate by $\Gamma^P{}_C(s)=\{a\in T^P{}_C\,|\exists s'\in S_C.s\xrightarrow{a}s'\}$,$\Gamma^H{}_C(s)=\{a\in T^H{}_C|\exists s'\in S_C.s\xrightarrow{a}s'\}$, and $\Gamma^O{}_C(s)=\{a\in T^R{}_C$ $|\exists s'\in S_C.\ s\xrightarrow{a}s'\}$) the subsets of input, internal and output actions that are enabled at the state s. Let $\Gamma_C(s)=\Gamma^P{}_C(s)\cup\Gamma^H{}_C(s)\cup\Gamma^R{}_C(s)$ be the subset of actions that are enabled at the state s.

Definition 2. An execution fragment of business component C is a finite alternating sequence of states and actions, denoted as: $\eta=s_0\xrightarrow{a_0}s_1\xrightarrow{a_1}s_2\ldots s_{n\text{-}1}\xrightarrow{a_{n-1}}s_n$, where, s_0 is the initial state, $s_i\xrightarrow{a_i}s_{i+1}\in T_C$ for all $0\le i<n$. If $s_n\in S_F$ is a final state, then η is called an execution.

Definition 3. Let $\eta=s_0\xrightarrow{a_0}s_1\xrightarrow{a_1}s_2\ldots s_{n\text{-}1}\xrightarrow{a_{n-1}}s_n$ be an execution fragment of business component C, the action sequence: $p(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}$ is called the trace of η. If η is an execution, then $p(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}$ is called a run or a word accepted by C. Let $L(C)$ be the language accepted by C.

Let $s\in S_C$ be a state of business component C, we say that s is reachable if there is an execution fragment whose last state is in s. we say that s is terminable if there exists an execution η such that s is on η. In the definition of behavior specification of business component, it is required that all states are reachable and terminable.

B. The Composition of Business Components

The composition of business components can be defined as a business component system that is composed of a set of business components and the dependency relationships among them. A dependency relationship is a connector between two business components that defines that one business component provides the operations that another business component requires.

Let C_i and C_j be two business components, the dependency relationship between C_i and C_j can be defined as $R(C_i,C_j)=\{(I,I')|I\in RI_i,I'\in PI_j,I\sqsubseteq I')\}$, where, RI_i is the set of required interfaces of C_i, PI_j is the set of provided interfaces of C_j, $I\sqsubseteq I'$ represents that I' can provide the operations that I requires. Let $S(C_i,C_j)$ represent the set of shared actions between C_i and C_j.

Definition 4. A business component system can be defined as $N=(CS,RS,BS)$, where, $CS=\{C_1,C_2,\ldots,C_n\}$ is the set of business components, $C_i=(n_i,PI_i,RI_i,BS_i)$, $BS_i=(S_i,\Sigma_i,T_i,s_{i0},S_{iF})$ $(0\le i\le n)$. $RS=\{R(C_i,C_j)|C_i,C_j\in C\}$is the set of dependency relationships between business components. BS is the behavior specification of N, it can be defined as the product of $C_1,C_2,\ldots,C_n$, denoted as: $BS=(S_N,\Sigma_N,T_N,s_{N0},S_{NF})$,where,

- $S_N=S_1\times S_2\times\ldots\times S_n$ is the set of composition states.
- $s_{N0}=(s_{10},s_{20},\ldots,s_{n0})\in S_N$ is the initial state.
- $S_{NF}=(\{s_{10}\}\cup S_{1F})\times(\{s_{20}\}\cup S_{2F})\times\ldots\times(\{s_{i0}\}\cup S_{2F})\backslash(s_{10},s_{20},\ldots,s_{n0})\in S_N$ is the set of final states.
- $\Sigma_N=\Sigma_N{}^P\cup\Sigma_N{}^H\cup\Sigma_N{}^R$ is the set of actions, where, $\Sigma_N{}^P=\bigcup_{i=1}^n\Sigma_i^P\backslash\bigcup_{0\le i,j\le n,i\ne j}S(C_i,C_j)$ is the set of input actions, $\Sigma_N{}^H=\left(\bigcup_{i=1}^n\Sigma_i^H\right)\cup\left(\bigcup_{0\le i,j\le n,i\ne j}S(C_i,C_j)\right)$ is the set of internal actions, and $\Sigma_N{}^R=\bigcup_{i=1}^n\Sigma_i^R\backslash\bigcup_{0\le i,j\le n,i\ne j}S(C_i,C_j)$ is the set of output actions.
- $T_N\subseteq S_N\times\Sigma_N\times S_N$ is the set of state transitions of business component system. If N satisfies any one condition as follows, it can transfer from state $s_N=(s_1,s_2,\ldots,s_n)\in S_N$ to state $s_N'=(s_1',s_2',\ldots,s_n')\in S_N$ by executing action $a\in\Sigma_N$:

(1) For an action $a\notin S(C_i,C_j)(1\le i,j\le n)$, there exists an input transition $s_i\xrightarrow{a}s_i'$ in C_i $(1\le i\le n)$, and for any j $(i\ne j,\ 1\le j\le n)$ such that $s_j=s_j'$.

(2) For an action $a\in S(C_i,C_j)(1\le i,j\le n)$, there exists an output transition $s_i\xrightarrow{a}s_i'\in T_i^R$ $(1\le i\le n)$, simultaneously there exists an input transition $s_j\xrightarrow{a}s_j'\in T_j^P$ $(i\ne j,1\le j\le n)$, and for any $k(k\ne i,j,1\le k\le n)$ such that $s_k=s_k'$.

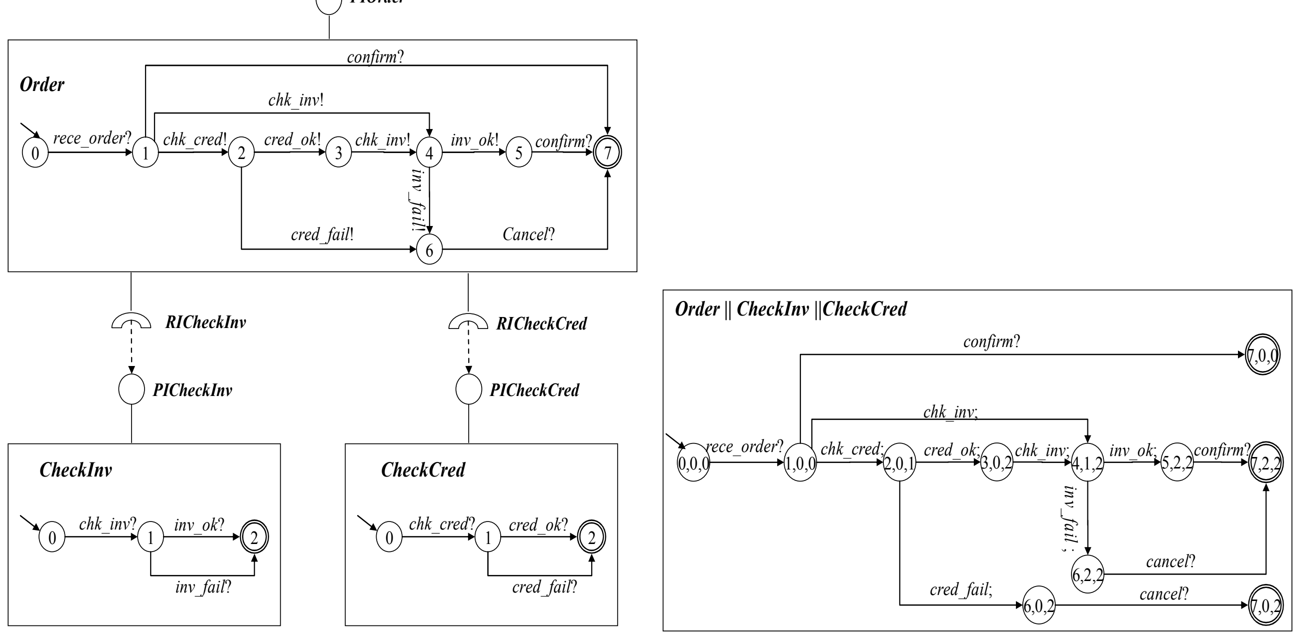
Since finite state machine is not necessarily input-enable in every state, in the behavior specification of business component system, one component may produce an output action that is an input action of another component in some composition state, but it not accepted, which means that the environment assumptions of two components with shared interface have mutual contradiction parts. These states are called as illegal composition states. In checking the compatibility of composition of business components, we should remove all reachable illegal composition states.

Definition 5. The set of illegal composition states of business component system N can be defined as: $Illegal(N)=\{(s_1,s_2,\ldots,s_n)\in S_N|\exists(s_i,s_j)(i\neq j,1\leq i,j\leq n),\exists a\in S(C_i,C_j),((a\in\Gamma^R{}_i(s_i)\wedge a\notin\Gamma^P{}_j(s_j))\vee(a\in\Gamma^R{}_j(s_j)\wedge a\notin\Gamma^P{}_i(s_i))\}$.

In the definition of the behavior specification of business component system, it is required that all composition states are compatible. We can construct the behavior specification which includes only compatible composition states by pruning the incompatible composition states, and removing any unreachable and blocking composition states.

C. A Practical Case

Figure 1 describes a business component system that includes three business components *Order*, *ChekInv* and *CheckCred*. The component *Order* consists of one provided interface: *PIOrder*, and two required interfaces: *RICheckInv* and *RICheckCred*, where, *PIOrder* includes three provided operations: receive orders (*rece_order*), confirm orders (*confirm*) and cancel orders (*cancel*), *RICheckInv* includes a required operation: check inventory (*chk_inv*) that is provided by the provided interface *PICheckInv* of *ChekInv*, and *RICheckCred* includes a required operation: check credit (*chk_cred*) that is provided by the provided interface *PICheckCred* of *CheckCred*.



(a) Business components *Order*, *CheckInv* and *CheckCred*.

(b) The behavior specification of composition of *Order*, *CheckInv* and *CheckCred*

Figure 1. Business component system

After component *Order* receives the orders send by a customer, it either checks the credit of the customer or skips this step and enters the step of checking inventory. Two possible return values of *chk_cred* are *cred_ok*, which indicates that the customer has a good credit, and *cred_fail*, which indicates that the customer has a poor credit. When component *Order* calls the operation *chk_cred* provided by *CheckCred*, if the return value is *cred_ok*, then the component continues to check inventory, otherwise, it cancels the orders. The two possible return values of *check_inv* are *inv_ok*, which indicates that the available stock can meet the requirement of the customer, and *inv_fail*, which indicates that the available stock can't meet the requirement of the customer. When the component *Order* calls operation *chk_inv* provided by *CheckInv*, if the return value is *inv_ok*, then it confirms the order, otherwise, it cancels the order.

Figure 1(a) illustrates the behavior specifications of components *Order*, *ChekInv* and *CheckCred*, and the dependency relation between them. Figure 1(b) illustrates the behavior specification of composition of *Order*, *ChekInv* and *CheckCred* that includes only compatible composition states.

IV. The composition of business components based on behavior and existence checks

A. The Process of business component retrieval

The process of business component retrieval can be divided into two phases: search phase and composition phase. Figure 2 illustrates the process of business component retrieval.

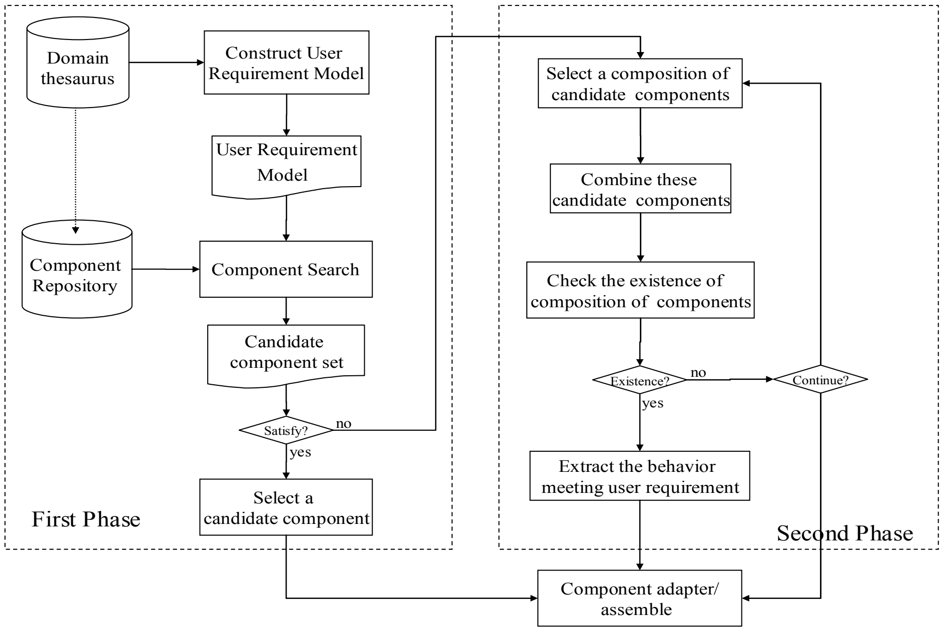

Domain thesaurus
Construct User Requirement Model
User Requirement Model
Component Repository
Component Search
Candidate component set
Satisfy?
no
yes
Select a candidate component
First Phase
Select a composition of candidate components
Combine these candidate components
Check the existence of composition of components
Existence?
no
Continue?
yes
Extract the behavior meeting user requirement
Second Phase
Component adapter/ assemble

Figure 2.The process of business component retrieval

In the search phase, at first, the user describes his requirement using behavior specification, and then uses corresponding search methods to retrieve a set of candidate business components that are equivalent, extension, compatible or weak compatible behavior specification of the user requirement [17]. To share the concepts of domain, when the user constructs the requirement model, he needs to refer to the domain thesaurus. In these candidate business components, if there are individual components that are equivalent or extension behavior specification of the user requirement, then he selects a component that is best close to the user requirement from these components as the final component, otherwise, enters the composition phase.

In the composition phase, when the number of candidate business components is very large, if we combine all candidate business components, then the complexity of computing the composition will be very high. To reduce the complexity, these candidate business components should be divided into several clusters. Components within the same cluster are related by equivalent or extension behavior specification relation. Two different clusters are related by non-behavior specification. The user can select one component from each cluster as a combined component. In general, the granularity of user requirement is not much bigger than that of candidate business components, so the number of clusters is not much larger, which can decrease the number of combined components. In the process of composition, these combined components should be also divided into several clusters. Components within the same cluster are related by dependency relationships. Two different clusters have not any dependency relationships. We first combine these components within the same cluster in a gradual fashion, and then continue to combine these clusters (each cluster can be regarded as a composite component). To educe the number of composition states, the incompatible composition should be pruned as early as possible. In this paper, we focus on checking the existence of composition of business components.

B. Behavior Specification Matching

Behavior specification matching aims at formalizing the matching relation between user requirement and the composition of a set of candidate business components. In this study, we use deterministic finite state machine whose syntax and semantic are similar to the behavior specification of business component to describe user requirement.

Definition 6. A user requirement can be defined as $R=(S_R,\Sigma_R,T_R,s_{R0},S_{RF})$, where, S_R is the set of finite states, Σ_R is the set of actions, $T_R\subseteq S_R\times\Sigma_R\times S_R$ is the set of state transitions, s_{R0} is the initial state, and S_{RF} is the set of final states.

Definition 7. Let $R=(S_R,\Sigma_R,T_R,s_{R0},S_{RF})$ be a user requirement, $N=(CS,RS,BS)$ be a business component system, where, $CS=\{C_1,C_2,\ldots,C_n\}$ is the set of business components, $C_i=(n_i,PI_i,RI_i,BS_i)$, $BS_i=(S_i,\Sigma_i,T_i,s_{i0},S_{iF})$ $(0\leq i\leq n)$. $RS=\{R(C_i,C_j)|C_i,C_j\in C\}$is the set of dependency relationships between business components. $BS=(S_N,\Sigma_N,T_N,s_{N0},S_{NF})$ is the behavior specification of N that includes only compatible composition states, if there exists a behavior specification matching relation from s_R to s_N: $\rho\subseteq S_R\times S_N$ such that satisfies the following conditions, then N is called as an extension behavior specification of R, denoted as $N\xrightarrow{M}R$.

1) $\forall s_R\in S_R,\ \exists s_N\in S_N.\ (s_R,s_N)\in\rho$;
2) $(s_{N0},s_{N0})\in\rho$;
3) $\forall s_{Nf}\in S_{NF},\exists\, s_{Nf}\in S_{NF}.\ (s_{Rf},s_{Nf})\in\rho$;
4) $\forall(s_R,s_N)\in\rho$ and $\forall\ a\in\Gamma_R(s_R),\ \exists a'\in\Gamma_N^{\,I}(s_N)\cup\Gamma_N^{\,H}(s_N).\ (a\sim a')\wedge(s_R,a,s_R')\in T_R\wedge(s_N,a',s_N')\in T_N^{\,P}\cup T_N^{\,H}\wedge(s_R',s_N')\in\rho$.

In the condition (4) above, $a\sim a'$ represents that a and a' is same or similar actions. To evaluate the similarity between actions, we need to refer to the domain thesaurus. $N\xrightarrow{M}R$ represents the behavior of N can meet completely the user requirement R. If $N\xrightarrow{M}R$, we can extract the behavior in accord with user requirement R from N.

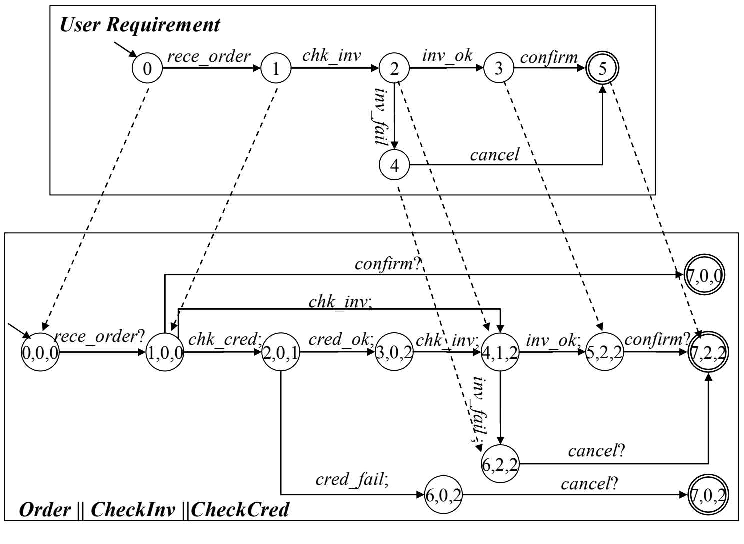


Figure 3.An example of behavior specification matching

Figure 3 describes an example of behavior specification matching between user requirement and the composition of business components, where, ρ={(0,(0,0,0)),(1,(1,0,0)), (2,(4,1,2)), (3,(5,2,2)), (4,(6,2,2)),(5,(7,2,2))}.

C. Behavior Mapping Graph

In this study, the behavior mapping graph from R to N that can be regarded as a finite state machine with inputs and outputs, is used to check the existence of composition of business components.

Definition 8. Let $R=(S_R,\Sigma_R,T_R,s_{R0},S_{RF})$ be a user requirement, $N=(CS,RS,BS)$ be a business component system, where, $CS=\{C_1,C_2,\ldots,C_n\}$ is the set of business components, $C_i=(n_i,PI_i,RI_i,BS_i)$, $BS_i=(S_i,\Sigma_i,T_i,s_{i0},S_{iF})$ $(0\le i\le n)$. $RS=\{R(C_i,C_j)|C_i,C_j\in C\}$ is the set of dependency relationships between business components. $BS=(S_N,\Sigma_N,T_N,s_{N0},S_{NF})$ is the behavior specification of N that includes only compatible composition states, the behavior mapping graph from R to N can be defined as: $M=(S_M,\Sigma_M,T_M,s_{M0},S_{MF})$, where,

- $S_M\subseteq S_R\times S_N$ is the set of mapping relations from S_R to S_N.
- $s_{M0}=(s_{R0},s_{N0})\in S_M$ is the mapping relation from s_{R0} to s_{N0}.
- $S_{MF}\subseteq S_{RF}\times S_{NF}$ is the set of mapping relations from S_{RF} to S_{NF}.
- $\Sigma_M=\Sigma_R\cup\Sigma_N$ is the set of actions.
- $T_M\subseteq S_M\times\Sigma_R\times\Sigma_N\times S_M$ is the set of mapping relations from T_R to T_N. We use symbol $s_M\xrightarrow{(a,a')}s_M'$ to represent the mapping relation from $s_R\xrightarrow{a}s_R'$ to $s_N\xrightarrow{a'}s_N'$, where, $s_M=(s_R,s_N)\in S_M$, $s_M'=(s_R',s_N')\in S_M$, $a\in\Sigma_R$, $a'\in\Sigma_N$, $(s_R,a,s_R')\in T_R$, $(s_N,a',s_N')\in T^P_N\cup T^H_N$, $a\sim a'$.

In the behavior mapping graph M, given a state mapping relationship $s_M=(s_R,s_N)\in S_M$, for a transition $s_R\xrightarrow{a}s_R'\in T_R$ in R, it can be mapped into more than one transition whose pre-state is s_N in N, let $T(s_M,a)=\{s_M\xrightarrow{(a,a')}s_M'|\exists s_N'\in S_N.\ s_N\xrightarrow{a'}s_N'\in T_N\wedge a\sim a'\}$ be the set of transition mapping relations from $s_R\xrightarrow{a}s_R'$ to T_N.

Definition 9. Let $s_M=(s_R,s_N)\in S_M$ be a mapping relationship from S_R to S_N, and $a\in\Gamma_R(s_R)$ is an action that is enabled in state s_R, if $T(s_M,a)=\phi\vee(s_R\in S_{RF}\wedge s_N\notin S_{NF})$, then s_M is called as an error mappings relationship. Let $ES=\{(s_R,s_N)|T((s_R,s_N),a)=\phi\vee(s_R\in S_{RF}\wedge s_N\notin S_{NF})\}$ be the set of error mappings relationships from S_R to S_N, $PreS(ES)=\{s_M|\exists s_M\xrightarrow{(a,a')}s_M'\in T_M.s_M'\in ES\}$ be the set of pre-states of ES, and $PostS(ES)=\{s_M'|\exists s_M\xrightarrow{(a,a')}s_M'\in T_M.s_M\in ES\}$ be the set of post-states of ES.

Definition 10. An execution fragment of behavior mapping graph M can be defined as: $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mn-1}\xrightarrow{(a_{n-1},a_{n-1}')}s_{Mn}$, where $s_{M0}=(s_{R0},s_{N0})$. If $s_{Mn}\in S_{MF}$, then η is called an execution of M. $p_I(\eta)=a_0,a_1,\ldots a_{n-1}$ is called the input action sequence on η, and $p_O(\eta)=a_0',a_1',\ldots,a_{n-1}'$ is called the output action sequence on η. If η is an execution, then $p_I(\eta)$ is called an input run on η, and $p_O(\eta)$ is called an output run on η. Let $L_I(M)$ be the set of input runs of M, and $L_O(M)$ be the set of output runs of M.

Theorem1. $L_I(M)\subseteq L(R)$, $L_O(M)\subseteq L(N)$.

Proof: If $L_I(M)=\phi$ and $L_O(M)=\phi$, then $L_I(M)\subseteq L(R)$,$L_O(M)\subseteq L(N)$, otherwise we need to prove that: (1) $p\in L_I(M)\Rightarrow p\in L(R)$; (2) $p'\in L_O(M)\Rightarrow p'\in L(N)$.

Let $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mn-1}\xrightarrow{(a_{n-1},a_{n-1}')}s_{Mn}$ be an arbitrary execution of M, where, $s_{Mi}=(s_{Ri},s_{Ni})$ $(0\le i\le n)$,$s_{Ri}\in S_R$, $s_{Ni}\in S_N$, $s_{M0}=(s_{R0},s_{N0})$, $s_{Mn}=(s_{Rn},s_{Nn})\in S_{MF}$. $p_I(\eta)=a_0,a_1,\ldots a_{n-1}\in L_I(M)$ is an input run on η, and $p_O(\eta)=a_0',a_1',\ldots,a_{n-1}'\in L_O(M)$ is an output run on η.

Let $\pi_R(\eta)=s_{R0}\xrightarrow{a_0}s_{R1}\xrightarrow{a_1}s_{R2}\ldots s_{Rn-1}\xrightarrow{a_{n-1}}s_{Rn}$ be the projection of η on R. According to the definition of behavior mapping graph, $s_{Ri-1}\xrightarrow{a_{i-1}}s_{Ri}$ $(0\le i\le n)$ is a transition of R, s_{R0} is the initial state of R, and $s_{Rn}\in S_{RF}$ is a final state of R, therefore $\pi_R(\eta)$ is an execution of R, and $p_I(\eta)$ is a run of R, that is, $p_I(\eta)\in L(R)$.

Similarly, let $\pi_N(\eta)=s_{N0}\xrightarrow{a_0'}s_{N1}\xrightarrow{a_1'}s_{N2}\ldots s_{Nn-1}\xrightarrow{a_{k-1}'}s_{Nn}$ be the projection of η on N. According to the definition of behavior mapping graph, $s_{Ni-1}\xrightarrow{a_{i-1}'}s_{Ni}$ $(0\le i\le n)$ is a transition of N, s_{N0} is the initial state of N, and $s_{Nn}\in S_{NF}$ is a final state of N, therefore $\pi_N(\eta)$ is an execution of N, and $p_O(\eta)$ is a run of N, that is, $p_O(\eta)\in L(N)$.

Thus, we have $L_I(M)\subseteq L(R)$and $L_O(M)\subseteq L(N)$. ■

D. Check the existence of composition of business components

In this section, we give an algorithm of checking the existence of composition of business components that only include compatible composition states.

Algorithm 1. Check the existence of composition of business components

Input: User requirement: $R=(S_R,\Sigma_R,T_R,s_{R0},S_{RF})$, business component system $N=(CS,RS,BS)$, where, $CS=\{C_1,C_2,\ldots,C_n\}$ is the set of candidate business components. RS is the set of dependency relationships between candidate business components. $BS=(S_N,\Sigma_N,T_N,s_{N0},S_{NF})$ is the behavior specification of N that only includes compatible composition states.

Output: T that represents $N\xrightarrow{M}R$, and F that represents $N\xrightarrow{NM}R$.

```
1  Construct the behavior mapping graph from $R$ to $N$: $M$.
2  Compute the set of error mappings relationships from $S_R$ to $S_N$: $ES$.
3  Let $ES_0=PreS(ES)\cup PostS(ES)$.
4  **Repeat: For** $k\geq 0$, let $ES_{k+1}=ES_k\cup PreS(ES_k)\cup PostS(ES_k)$.
5  **Until:** $ES_{k+1}=ES_k$.
6  Let $S_{SM}=S_M/ES_k$ and $S_{SMF}=S_{MF}/ES_k$.
7  Let $T_{SM}=\{s_M\xrightarrow{(a,a')}s_M'|s_M, s_M'\in S_{SM}\}$.
8  **If** $s_{M0}\in S_{SM}$ **Then**
9      **Output** $SM=(S_{SM},\Sigma_M,T_{SM},s_{M0},S_{SMF})$ that is a subgraph of $M$.
10     **Return** $T$.
11 **Else Return** $F$.
```

In algorithm 1, the computational complexity of constructing behavior mapping graph is $O(|S_R|^2|S_N|^2)$, and the computational complexity of checking the existence of composition of business components is $O(|S_R|^2|S_N|^2)$ in the worst case and $O(|S_R||S_N|)$ in the best case. In the following we prove the correction of algorithm 1.

In order to prove the correction of algorithm 1, we need to prove theory 2 by giving two lemmas.

Lemma 1. If $s_{M0}\in S_{SM}$, then $L(R)=L_l(SM)$,else $L(R)\neq L_l(SM)$.

Proof: If we can prove $L_l(SM)\subseteq L(R)\wedge L(R)\subseteq L_l(SM)$, then we can conclude $L(R)=L_l(SM)$.

Since SM is a subgraph of M, $L_l(SM)\subseteq L_l(M)$, according to theorem 1, we have $L_l(M)\subseteq L(R)$, therefore $L_l(SM)\subseteq P(R)$. In the following, we prove only $L(R)\subseteq L_l(SM)$.

Let $\eta_R=s_{R0}\xrightarrow{a_0}s_{R1}\xrightarrow{a_1}s_{R2}\ldots s_{Rn\text{-}1}\xrightarrow{a_{n-1}}s_{Rn}$ be an arbitrary execution of SR, $p=a_0,a_1,\ldots,a_{n\text{-}1}\in L(SR)$ is the run on η_R, if we can prove that $p\in L(SM)$, then we can conclude $L(R)\subseteq L_l(SM)$.

In the following we use mathematical induction to prove that there exists an execution $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mn\text{-}1}\xrightarrow{(a_{n-1},a_{n-1}')}s_{Mn}$ in SM such that $p_l(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}=p$, where, $s_{Mi}=(s_{Ri},s_{Ni})(0\leq i\leq n)$.

(1)We need to prove that i=1 is true, which means that there exists an execution fragment $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}$ in SM such that $p_l(\eta)=a_0$, where $s_{Mj}=(s_{Rj},s_{Nj})(0\leq j\leq 1)$. We use reduction to absurdity, suppose $T(s_{M0},a_0)=\phi$.

Since $T(s_{M0},a_0)=\phi$, it must be the case that s_{M0} is an illegal mapping relationship from S_R to S_N. According algorithm 1, s_{M0} should be removed from M, that is, $s_{M0}\notin S_{SM}$, which contradicts our premise that $s_{M0}\in S_{SM}$. Therefore, there exists an execution fragment $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}$ in SM such that $p_l(\eta)=a_0$, where, $s_{Mj}=(s_{Rj},s_{Nj})(0\leq j\leq 1)$.

(2)Suppose that $i=k$ is true, which means that there exists an execution fragment $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mk\text{-}1}\xrightarrow{(a_{k-1},a_{k-1}')}s_{Mk}$ in SM such that $p_l(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}$, where, $s_{Mj}=(s_{Rj},s_{Nj})(0\leq j\leq k)$. We prove that $i=k$+1 is true, which means that we need to prove $T(s_{Mk},a_k)\neq\phi$. We use reduction to absurdity, suppose $T(s_{Mk},a_k)=\phi$.

Since $T(s_{Mk},a_k)=\phi$, it must be the case that s_{Mk} is an illegal mapping relationship from S_R to S_N. According to algorithm 1, s_{Mk} should be removed from M, that is, $s_{Mk}\notin S_{SM}$, which contradicts our assumption in $i=k$. Therefore there exists a fragment $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mk\text{-}1}\xrightarrow{(a_{k-1},a_{k-1}')}s_{Mk}\xrightarrow{(a_k,a_k')}s_{Mk+1}$ in SM such that $p_I(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}$ where, $s_{Mj}=(s_{Rj},s_{Nj})(0\leq j\leq k+1)$.

According to the proof above, $\eta=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mn\text{-}1}\xrightarrow{(a_{n-1},a_{n-1}')}s_{Mn}$ is an execution fragment of SM. In the following we prove that $p_I(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}$ is a run of SM, which means that we need to prove that $s_{Mn}=(s_{Rn},s_{Nn})\in S_{SMF}$. We use reduction to absurdity, suppose $s_{Mn}=(s_{Rn},s_{Nn})\notin S_{SMF}$.

Since $s_{Mn}=(s_{Rn},s_{Nn})\notin S_{SMF}$ and $s_{Rn}\in S_{RF}$, it must be the case that $s_{Nn}\notin S_{NF}$. Therefore $s_{Mn}=(s_{Rn},s_{Nn})$ is an illegal mapping relationship from S_R to S_N, according to algorithm 1, s_{Mn} should be removed from M, that is, $s_{Mn}\notin S_{SM}$, which contradicts our assumption that $s_{Mn}\in S_{SM}$, therefore, $s_{Mn}=(s_{Rn},s_{Nn})\in S_{SMF}$, which means $p=p_I(\eta)=a_0,a_1,\ldots,a_{n\text{-}1}\in L_I(SM)$. That is $L(R)\subseteq L_I(SM)$.

If $s_{M0}\notin S_{SM}$, then $L_I(SM)=\phi$, but $L(R)\neq\phi$, thus $L(R)\neq L_I(SM)$. ■

Lemma 2. $L(R)=L_I(SM)\Leftrightarrow N\xrightarrow{M}R$.

Proof: ($\Rightarrow$) If we can prove that $S_{SM}\subseteq S_R\times S_N$ satisfies the behavior specification matching relation from S_R to S_N, then we can conclude that $N\xrightarrow{M}R$. According to definition 7, we need to prove the four conditions as follows:

(1) $\forall s_R\in S_R,\ \exists s_N\in S_N\ .(s_R,s_N)\in S_{SM}$.

We use reduction to absurdity, suppose that $\exists s_{Rk}\in S_R,\ \forall s_N\in S_N.\ (s_{Rk},s_N)\notin S_{SM}$.

Let $\eta_R=s_{R0}\xrightarrow{a_0}s_{R1}\xrightarrow{a_1}s_{R2}\ldots s_{Rk}\xrightarrow{a_k}s_{Rk+1}\ldots s_{Rn\text{-}1}\xrightarrow{a_{n-1}}s_{Rn}$ be an execution that includes state s_{Rk}. $p(\eta_R)=a_0,a_1,\ldots,a_k,\ldots,a_{n\text{-}1}$ is run on η_R. Since $\forall s_N\in S_N.(s_{Rk},s_N)\notin S_{SM}$, it must be the case that there doesn't exist a run $\eta_{SM}=s_{M0}\xrightarrow{(a_0,a_0')}s_{M1}\xrightarrow{(a_1,a_1')}s_{M2}\ldots s_{Mk}\xrightarrow{(a_k,a_k')}s_{Mk+1}\ldots s_{Mn\text{-}1}\xrightarrow{(a_{n-1},a_{n-1}')}s_{Mn}$ that includes state s_{Mk} in SM such that $s_{Mi}=(s_{Ri},s_{Ni})\in S_{SM}$, $(0\leq i\leq n)$, therefore $p_I(\eta_{SM})=a_0,a_1,\ldots,a_k,\ldots,a_{n\text{-}1}\notin L_I(SM)$, but $p(\eta_{SR})=a_0,a_1,\ldots,a_k,\ldots,a_{n\text{-}1}\in L(SR)$, thus $L(R)\neq L_I(SM)$, which contradicts our assumption, therefore $\forall s_R\in S_R,\exists s_N\in S_N.\ (s_R,s_N)\in S_{SM}$.

(2) $(s_{R0},s_{N0})\in S_{SM}$.

We use reduction to absurdity, suppose that $(s_{R0},s_{N0})\notin S_{SM}$.

Since $(s_{R0},s_{N0})\notin S_{SM}$, $L_I(SM)=\phi$. Because $L(R)\neq\phi$, it must be the case that $L_I(SM)\neq L(R)$, which contradicts our assumption that $(s_{R0},s_{N0})\notin S_{SM}$, thus $(s_{R0},s_{N0})\in S_{SM}$.

(3) $\forall s_{Rf}\in S_{RF},\ \exists s_{Nf}\in S_{NF}.\ (s_{Rf},s_{Nf})\in S_{SM}$.

We use reduction to absurdity, suppose that $\exists s_{Rf}\in S_{RF},\ \forall s_{Nf}\in S_{NF}.\ (s_{Rf},s_{Nf})\notin S_{SM}$.

Since $\forall s_{Nf}\in S_{NF}.(s_{Rf},s_{Nf})\notin S_{SM}$, according to the result of (1), there exists a state $s_N\in S_N\backslash S_{NF}$ such that $(s_{Rf},s_N)\in S_{SM}$. Since $s_{Nf}\in S_{RF}$ and $s_N\notin S_{NF}$, and hence $(s_{Rf},s_N)\notin S_{SMF}$, according to algorithm 1, $(s_{Rf},s_N)\notin S_{SM}$, which contradicts our assumption that $(s_{Rf},s_N)\in S_{SM}$, therefore, $\forall s_{Rf}\in S_{RF},\ \exists s_{Nf}\in S_{NF}.\ (s_{Rf},s_{Nf})\in S_{SM}$.

(4) $\forall(s_R,s_N)\in S_{SM}$ and $\forall a\in\Gamma_R(s_R),\ \exists a'\in\Gamma_N^I(s_N)\cup\Gamma_N^H(s_N).\ (a\sim a')\wedge(s_R,a,s_R')\in T_R\wedge(s_N,a',s_N')\in T^P_N\cup T^H_N\wedge(s_{SR}',s_N')\in S_{SM}$.

Since there doesn't exist illegal mapping relationships in SM, $\forall(s_R,s_N)\in S_{SM}$ and $\forall a\in\Gamma_R(s_R)$, $\exists a'\in\Gamma_N^I(s_N)\cup\Gamma_N^H(s_N).\ (a\sim a')\wedge(s_R,a,s_R')\in T_R\wedge(s_N,a',s_N')\in T_N\wedge(s_{SR}',s_N')\in S_{SM}$.

According to the proof above, we have $N\xrightarrow{M}R$.

($\Leftarrow$) Because $N\xrightarrow{M}R$, there exists a behavior specification relation $\rho\subseteq S_{SR}\times S_N$ from SR to N, let SM be a behavior mapping subgraph that is composed of the states in ρ and the transitions between them, because SM doesn't include illegal mapping relationships from S_R to S_N, and $s_{M0}\in\rho$, according to lemma2, $L(R)=L_I(SM)$. ■

Theorem 2. Let R be a user requirement, and N be a business component system, M is the behavior graph from R to N, s_{M0} is the initial state of M, SM is a subgraph of M that is constructed according to algorithm 1, if $s_{M0}\in S_{SM}$, then $N\xrightarrow{M}R$, else $N\xrightarrow{NM}R$.

According to lemma 1 and lemma 2, we can prove easily theorem 2. According to theorem 2, if we want to justify whether N can meet R, we only need to justify after we remove all error mapping relationships and corresponding states from M, whether s_{M0} is still exists, if it exists, then $N \xrightarrow{M} R$, else $N \xrightarrow{NM} R$ that represents that the behavior of N can't completely meet the behavior specified by R.

F. Extract the Behavior Satisfying User Requirement from Behavior Mapping Graph

If $N \xrightarrow{M} R$, we can extract the behavior is accord with user requirement from the behavior mapping subgraph SM. Given a state $s_{SM}=(s_R,s_N)\in S_{SM}$, for a transition $s_R \xrightarrow{a} s_R'$ in R, it can be mapped into more than one input or internal transition in N, i.e., $|T(s_{SM},a)|>1$. Because R is a deterministic finite state machine, we should choice a transition from $T(s_{SM},a)$, and then delete other transitions, after we delete redundant transitions, there maybe exist many unreachable states, we continue to delete these states and corresponding transitions from SM until there aren't unreachable states in SM, let SSM be the subgraph of SM that doesn't include illegal states and unreachable states. Based on SSM , we can extract the behavior satisfying user requirement from N. In the following we give the method extracting the behaviors.

(1) For each state $s_{SSM}=(s_R,s_N)$ in SSM, omit the state of user requirement s_R, therefore s_{SSM} becomes s_N.

(2) For each transition $s_N \xrightarrow{(a,a')} s_N'$ in SSM, omit the action of user requirement a, $s_N \xrightarrow{(a,a')} s_N'$ becomes $s_N \xrightarrow{a'} s_N'$.

(3) After we omit all states of user requirement, if there are homonymous states in SSM, we merge these states into one state that called as composition state. For each merged state s_N, if there exists transition $s_N \xrightarrow{a'} s_N'$ ($s_N' \xrightarrow{a'} s_N$), then

a) if $s_N=s_N'$, substitute s_N and s_N' with composition state;

b) if $s_N\neq s_N'$, substitute s_N with composition state.

(4) Merge all homonymous transitions into one transition.

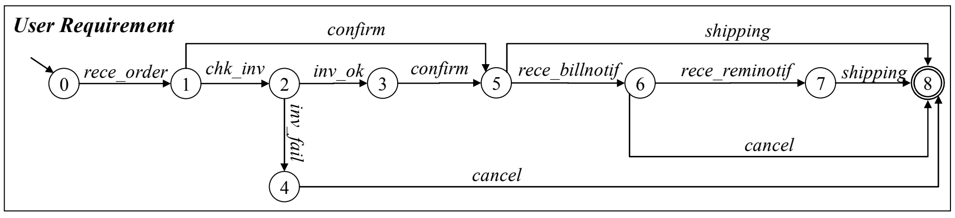
According to the method above, we can abstract the behavior in accord with user requirement from business component system.

V. Example of application

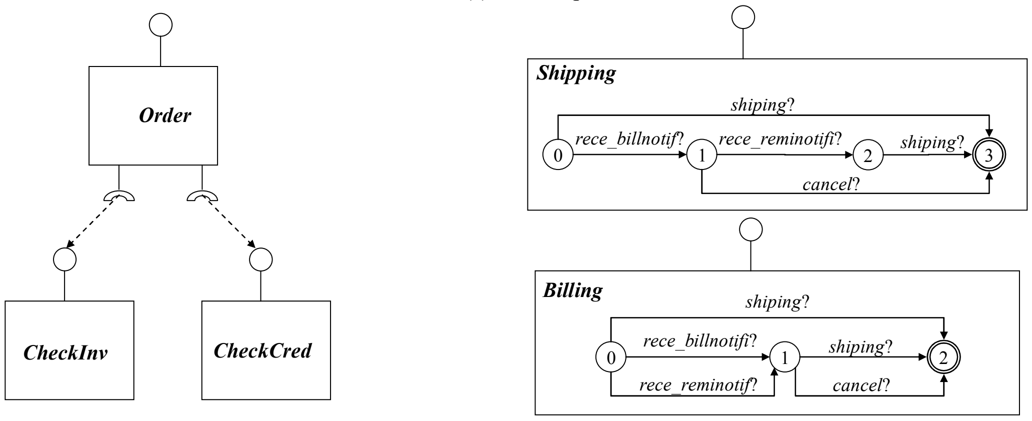
In this section, we give an example to explain the process of checking the existence of composition of business components and the method of extracting the behaviors in accord with user requirement from these business components.

Figure 4(a) describes a user requirement R, and figure 4(b) describes a composition that consists of five candidate components: *Order*, *CheckInv*, *CheckCred*, *Shipping* and *Billing*, where, the behavior specifications of *Order*, *CheckInv* and *CheckCred* are shown as figure 1, and the behavior specifications of *Shipping* and *billing* are shown as the right side of figure 4(b).

Figure 5 shows the behavior mapping graph from R to the composition of *Order*, *CheckInv*, *CheckCred*, *Shipping* and *Billing*. Because $T((6,(7,2,2,1,0),rece\_reminotif)=\phi$, (6,(7,2,2,1,0)) is a error mapping relationship. In (6,(7,0,0,0,0)), for transition: $5 \xrightarrow{shipping} 8$ in R, there are two possible mapping relationships. According to the approach proposed in section IV, we can extract the behavior in accord with user requirement (shown as figure 6) from the behavior mapping graph shown as figure 5.



(a) User Requirement



(b) Candidate Business Components

Figure 4. User requirement and candidate components

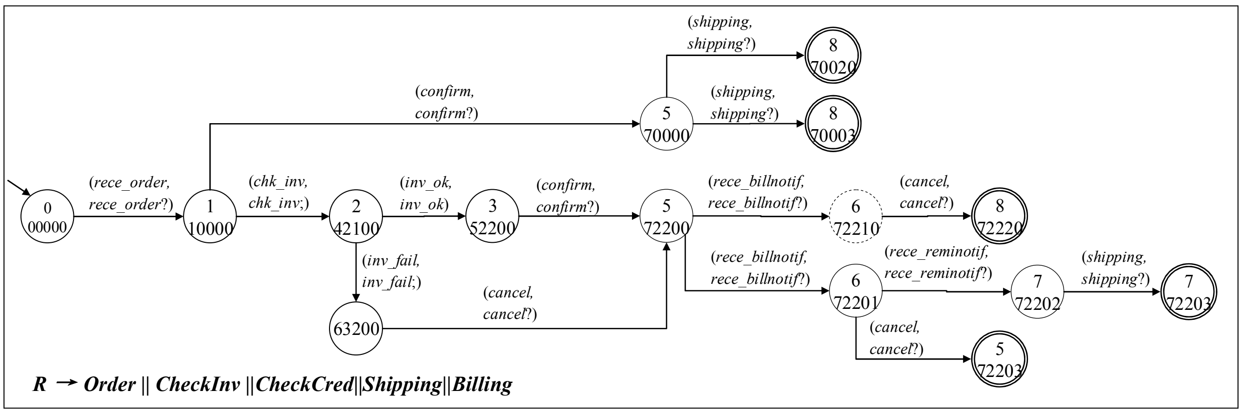


Figure 5. Behavior mapping graph

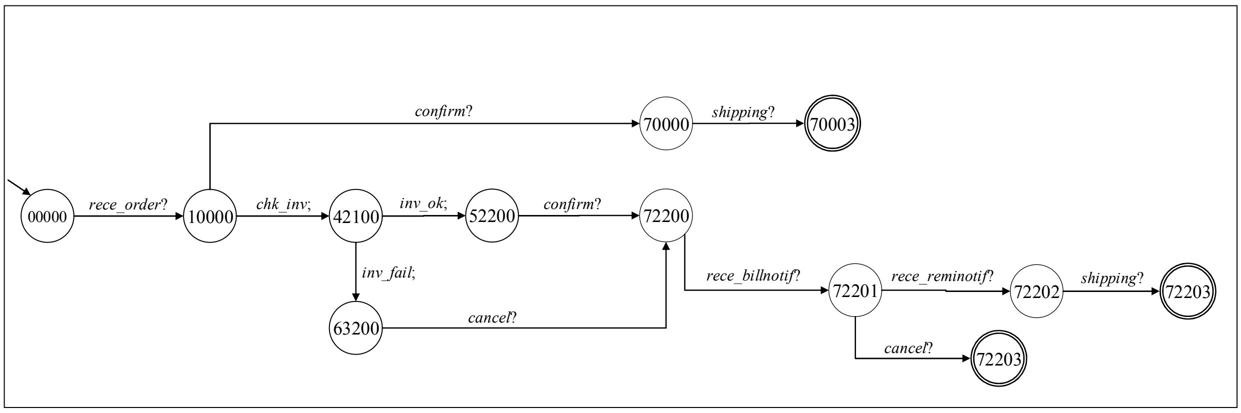


Figure 6. Behavior specification in accord with user requirement

VI. Conclusion

This paper proposes a method of checking the existence of composition of components based on behavior specification matching. We use determine finite state machine that extends the interface automata to model behavior specifications of business component, and use the product of deterministic finite state machine describe the composition of business components whose behavior can be regarded as a nondeterministic finite state machine. The behavior mapping graph is used to check the existence of composition of components and extracting the behavior in accord with user requirement. In the future, we intend to study more ways of composition of business components.

Acknowledgements

This paper is supported by the National Natural Science Foundation of China under Grant No.60673025 and the National High-Tech Research and Development Plan of China under Grant No.2006AA04Z150

References

[1] Alfaro L, Henzinger TA. Interface automata. In: Wermelinger M, Gall H, eds. Proc. of the 9th Annual ACM Symp. on Foundations of Software Engineering (FSE 2001). New York: ACM Press, 2001. 109-120.

[2] Colin Blundell, Kathi Fisler, Shriram Krishnamurthi, Pascal Van Hentenryck. Parameterized Interfaces for Open System Verification of Product Lines. 19th IEEE International Conference on Automated Software Engineering (ASE'04) 2004. pp.258-267.

[3] Osamu Shigo, Atsushi Okawa, Daiki Kato. Constructing Behavioral State Machine using Interface Protocol Specification, XIII Asia Pacific Software Engineering Conference (APSEC'06). IEEE Computer Society, December 2006, pp.191-198.

[4] By D.C. Craig, W.M. Zuberek. Compatibility of Software Components - Modeling and Verification, International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX'06). IEEE Computer Society, May 2006, pp.11-18.

[5] Nabil Hameurlain. A Formal Framework for Component Protocols Behavioural Compatibility. Proceedings of the XIII Asia Pacific Software Engineering Conference, IEEE Computer Society, Washington, DC, USA , 2006, pp.87-94.

[6] N. Hameurlain. Formalizing Compatibility and Substitutability of Role- based. 4th International/Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2005, Lecture Notes in Computer Science, Springer-Verlag, LNAI/LNCS Vol. 3690, 2005, pp 153-162.

[7] A. M. Zaremski, J. M. Wing. “Specification Matching of Software Components”. ACM Transactions of Software Engineering and Methodology, 1997,6 (4): 333-369.

[8] Bracciali A, Brogi A, Canal C. A formal approach to component adaptation. Joural of Systems and Software, 2005,74(1):45-54.

[9] C. Canal, E. Pimentel, and J.M. Troya. “Compatibility and Inheritance in Software Architectures” Science of Computer Programming, 41(2):105-138. 2001.

[10] Frantisek Plasil, Stanislav Visnovsky. Behavior Protocols for Software Components. IEEE Transactions on Software Engineering, 2002, 28(11)：1056-1076.

[11] B.H.C. Cheng and J.J.Jeng. Reusing analogous components. IEEE Transaction on Knowledge and Data Engineering, 9(2), March, 1997.

[12] Redondo, R.P.D.; Arias, J.J.P.; Vilas, A.F.; Martinez, B.B. Approximate Retrieval of incomplete and formal specifications applied to vertical reuse. Proceedings of International Conference on Software Maintenance (ICSM’02), 3-6 Oct. 2002:618- 627.

[13] Hai Zhuge. An inexact model matching approach and its applications. The Journal of Systems and Software 67 (2003) 201–212.

[14] Praphamontripong, U.; Hu, G. XML-based software component retrieval with partial and reference matching. Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration，8-10 Nov. 2004:127 – 132.

[15] Mili, R. Mili, and R. Mittermeir, Storing and Retrieving Software Component: A Refinement Based Approach. IEEE Transactions on software Engineering, 1999, 23(7), pp. 139-170.

[16] Somjit Arch-int, Dentcho N. Batanov. Development of industrial information systems on the Web using business components. Computer in Industry 2003,50(2):231-250.

[17] Fanchao Meng, Dechen Zhan, Xiaofei Xu. A web service retrieval method based on behavior specification matching. System and Information Science Notes, 2007,1(4),pp.402-408.

Fanchao Meng is a Ph.D candidate at School of Computer Science and Technology in Harbin Industrial of Technology(HIT),China. His current research areas include software engineering, Model Driven Architecture(MDA), software reuse and reconfiguration.

Dechen Zhan is a professor in School of Computer Science and Technology at Harbin Institute of Technology (HIT), China. His research interests include computer integrated manufacturing system (CIMS), enterprise resource planning(ERP), decision support systems (DSS), Model Driven Architecture(MDA), software reuse and reconfiguration, etc.

XiaoFei Xu is a professor and dean of School of Computer Science andTechnology at Harbin Institute of Technology (HIT), China. His research interests include computer integrated manufacturing system (CIMS), management and decision information system, software engineering, etc.