

A Formal Model for Abstracting the Interaction of Web Services

Li Bao

Institute of Software Engineering, Dalian Maritime University, Dalian, China
Email: ebond@163.com

Weishi Zhang and Xiong Xie

Institute of Software Engineering, Dalian Maritime University, Dalian, China
Email: {teesiv, xxyj}@dlmu.edu.cn

Abstract—This paper addresses the problems of modeling the interaction of Web services when they are composed together. Many subtle errors such as message not received and deadlock may occur due to uncontrolled concurrency of Web services. A model called IMWSC (Interaction Module for Web Service Composition, IMWSC for short) is proposed. The proposed model is used to abstract and analyze the interaction of web services. IMWSC is given a formal semantics by means of CCS (Calculus of Communicating System, CCS for short), which is a kind of process algebra that can be used to model concurrent systems. The application of this model is further investigated in a case study. Some important points related to verify the correctness of interaction of Web service are discussed.

Index Terms—Web Service, Interaction, Formal Method, IMWSC

I. INTRODUCTION

In order to survive the massive competition created by the new online economy, many organizations are rushing to put their core business competencies on the Internet as a collection of web services for more automation and global visibility^[1]. The concept of web service has become recently very popular. Web services are software applications which can be used through a network (intranet or Internet) via the exchange of messages based on XML standards^[2]. It has become a vehicle of web services rather than just a repository of information.

The ability to efficiently and effectively share services on the Web is a critical step towards the development of the new online economy driven by the Business-to-Business (B2B) e-commerce^[1]. Existing enterprises would form alliances and integrate their services to share costs, skills, and resources in offering a value-added service to form what is known as composite service.

A composite web service is a system that consists of several conceptually autonomous but cooperating units. In order to establish a long-running service composition,

many languages and tools emerged, which provide different schemas to glue service operations properly. Service composition approaches can be generally divided into two categories^[3, 4]: business flow based approach and semantic based approach. Some famous projects on web service are based on business flow^[2, 4], such as eFlow^[5], METEOR-S^[6], SELF-SERV^[7]; Semantics based approach composes services based on ontology and relies on the use of AI planning techniques to automatically search, orchestrate, compose and execute services. Representative projects on web service research that is based on Semantics are: WebDG^[8], SWORD^[9], SHOP^[10].

From a software engineering viewpoint, the construction of new services by the static or dynamic composition of existing services raises exciting new perspectives which can significantly impact the way industrial applications will be developed in the future — but they also raise a number of challenges. Among them is the essential problem of guaranteeing the correct interaction of independent, communicating software pieces^[2].

One legitimate question is therefore whether or not the correct and reliable interaction of web services can be guaranteed to a great extent by introducing the formal description techniques. Our investigations suggest a positive answer. This paper addresses the problem of formally modeling the interaction of web services when they are composed together, be it in a dynamic or static way. A model for abstracting and analyzing one scenario of the interaction process of web services called IMWSC is proposed. After the interaction of web service is described in an abstract way, available supporting tool can be used to determine whether or not this interaction process satisfies the desired properties which are expressed in a kind of modal logic.

This paper is structured as follows. Section 2 discusses the related work. In Section 3, we present IMWSC. Section 4 defines the semantics of IMWSC. The application of IMWSC is investigated in a case study in Section 5. And the conclusion and future work are drawn up in Section 6.

II. Related Work

Petri nets are a formal model for concurrency. Since the semantics of Petri nets is formally defined, by mapping each BPEL process to a Petri net a formal model of BPEL can be obtained which allows the verification techniques and tools developed for Petri nets to be exploited in the context of BPEL processes. Many works such as [11, 12, 21, 22] introduce the Petri net based method for describing and verifying web service.

In [21], Schmidt and Stahl discuss a mapping from BPEL to Petri nets by giving several examples. Each BPEL construct is mapped into a Petri net pattern.

In [22], Schlingloff, Martens and Schmidt also consider the usability problem. They show that usability can be expressed in alternating-time temporal logic. As a consequence, model checking algorithms for this logic can be exploited to check for usability.

As research aiming at facilitating web services integration and verification, WS-Net introduced in [11] is an executable architectural description language incorporating the semantics of Colored Petri-net with the style and understandability of object-oriented concepts.

In [12], Tao provide a web service composition model which is based on a kind of advanced Petri-net, OOPN (Object Oriented Petri Net). A web service can be mapped to an OOPN system based on this model and different OOPN system can be integrated together into a composite service via message passing.

A process algebra is a rather small concurrent language that abstracts from many details and focuses on particular features. There are several relevant publications [1, 13, 14] for process algebra based methods.

Gwen Salaün, Lucas Bordeaux present an overview of the applicability of process algebras in the context of web services in [1].

Authors present a framework for the design and the verification of WSs using process algebras and their tools in [13].

Li Bao, Weishi Zhang present a CCS based method for describing and verifying the behaviour of web service in [14].

III. Defining IMWSC

A. Initiative of IMWSC

For the Petri net based methods, one major defect is that the number of the places and the transitions described in a Petri net is too large. Researchers often map each element in a web service composition language to an element in Petri net and do not restrict the number of the places and the transitions in a Petri net. If the number of the places and the transitions described in a Petri net is not restricted, the designers will meet a condition of state explosion, which is very difficult to be dealt with; another major defect for the Petri net based methods is the lack of the description of interaction process of web services. The Petri net based methods often put their emphasis on describing the workflow inside a web service, and do not present the complicate interaction process of web services.

For the process algebra based methods, one major defect is that some kinds of complex structure of web service composition can not be defined by using these methods; another major defect for the process algebra based methods is that the lack of rigorous translation mechanism between the element of web service composition language and the element of process algebra. These methods often give simple corresponding relations and translation rules. These relations and rules can not guarantee the correct reservation of the information related to behavior and are apt to lead to the loss of information. We adopt a kind of hierarchically refined description method to define the interaction process of web services, i.e., we divide the interaction process of web services into smaller parts, which is defined as Interaction Module for Web Service Composition (IMWSC in short). For each of these parts, a scenario of the interaction of web services is defined. These smaller parts, i.e., modules, have a common property that the outcome of each module is determinate, in other words, each of the module has only one terminative state. This important property suggests these modules can be composed. Therefore, by mapping each module to a transition in a Petri net, modules which describe the scenarios of the interaction of web services are strictly composed. However, for the limit of the length, we only introduce the definition and properties of a module, i.e. IMWSC model, method about how to compose these modules will be introduced in further work.

Instead of composing activities of web service, we compose modules. The merit of our approach is that it can effectively reduce the number of the objects to be analysis, such that the interaction process of web service is described more concisely, as well as the state explosion can be avoided. At the same time, web service composition with complex structure can be described by composing these modules, while the process algebra based methods can not achieve.

Another benefit of our approach is the introducing of the semantic of IMWSC. The semantic of IMWSC comprises three parts: semantic domain, semantic range, and valuation function. A process calculus CCS (Calculus of Communicating Systems, CCS in short) [15, 16] is introduced as a semantic range, and then valuation functions are defined that translate an IMWSC (semantic domain) into a process term. Since the valuation functions are rigorously defined, the correct reservation of the information related to behavior can be guaranteed, such that the loss of information can be avoided.

B. Formal Definition of IMWSC

A web service is a software application which can be used through a network (intranet or Internet). For a web service, the basic functional unit is operation. The process of web service invocation is actually the process of operation invocation. For IMWSC, the invocation of operation is modeled by *Activity*. For a better control of the structure of activities, we introduce a set of processes, i.e. *Proc*, as the basic control unit. A process, i.e. an element in *Proc*, is a linear concatenation of activities. If

the output data of one operation opr_1 is the input data of another operation opr_2 , we consider that there is a corresponding relation between opr_1 and opr_2 . For IMWSC, we introduce the binary relation R_a to represent this kind of relation. Symbol L is introduced into IMWSC to record the interaction history of web services. We described the interaction process of web services in a scenario in the way defined by IMWSC, in other words, the definition of an instance of IMWSC is the definition of the interaction process of web services in a scenario.

Definition 1. (IMWSC) Formally, an IMWSC is a septuple $\langle Service, Proc, Activity, L, Message, R_a, \mathcal{F} \rangle$, where:

- *Service* denotes a set of web services;
- *Proc* is a set of processes ;
- *Activity* is a set of activities ;
- L is a set of sequences of activities;
- *Message* is a set of messages that are exchanged by services;
- $R_a \subseteq Activity \times Activity$ is a binary relation;
- \mathcal{F} is a sextuple $\langle f_{pT}, f_{pS}, f_{pU}, f_{aP}, f_{aT}, f_{mA} \rangle$, where:
 - $f_{pT} : Proc \rightarrow \{c, b\}$ is a mapping that describes the type of each process (*composite* or *basic*);
 - $f_{pS} : Proc \rightarrow Service$ is a mapping that describes the type of each process (*composite* or *atomic*);
 - $f_{pU} : Proc \rightarrow Proc$ is a mapping that associates a process with a composite process;
 - $f_{aP} : Activity \rightarrow Proc$ is a mapping that associates each activity with a process ;
 - $f_{aT} : Activity \rightarrow \{ii, io, ei, eo, ex\}$ is a mapping that describes the type of each activity (*internal input, internal output, environmental input, environmental output, execute*);
 - $f_{mA} : Message \rightarrow Activity$ is a mapping that associates each message with an Activity.

We let $f_{con}(proc) = \{a \mid a \in Activity \wedge f_{ap}(a) = proc\}$ for $p \in Proc \wedge f_{pT}(p) = b$; Let $\langle_c \subseteq Activity \times Activity$ be an partial order relation over *Activity*, defined as: $\langle_c = \{ (a_1, a_2) \mid a_1, a_2 \in Activity \wedge f_{ap}(a_1) = f_{ap}(a_2) \wedge (a_1 \text{ happens earlier than } a_2) \}$; An element $proc$ in *Proc* is constructed by the following grammar:

$$proc = \alpha \mid proc_1 \parallel proc_2 \mid proc_1 \prec proc_2,$$

where: $\alpha \in Activity$; $proc1, proc2 \in Proc$.

- $proc_1 \parallel proc_2$ is a new process that performs $proc1$ and $proc2$ independently;
- $proc_1 \prec proc_2$ is a new process that performs $proc1$ and $proc2$ sequentially.

Fig. 1 presents an illustration of the structure of IMWSC. In Fig.1, a service is visualized by a circle; interaction of services is visualized by a pair of parallel arrows (with opposite directions); the interaction process Definition, i.e., the definition of an instance of IMWSC, is visualized by a rectangle.

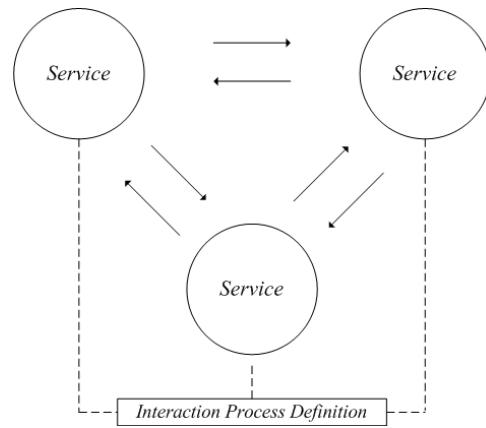


Figure 1. Structure of IMWSC

C. The Necessary Condition for the Correctness of IMWSC

The fundamental requirement for a correct interaction process of services is that each input of a service shall be met by another service. Thus the basic requirement that guarantees the correctness of IMWSC is:

for any activity $a_1 \in Activity$, if it is an input activity, then, there shall be another activity $a_2 \in Activity$, such that $a_1 R_a a_2$. The necessary condition for the correctness of IMWSC can also be defined as the following predicative formula:

$$\forall a \in Activity ((f_{aT}(a) = ii \vee f_{aT}(a) = io) \rightarrow (\exists a' \in Activity (a R_a a' \vee a' R_a a)))$$

IV. Formal Semantics of IMWSC

Formal semantic descriptions of a model are the basis for proving properties of this model. Moreover, they provide precise documentation of model design and standards for implementations, and (sometimes) they can be used for generation of prototype implementations.

The formal semantics of IMWSC comprises three parts: semantic domain, semantic range, and valuation function. A process calculus CCS (Calculus of Communicating Systems, CCS for short) is introduced as a semantic range,

and then valuation functions are defined that translate an IMWSC (semantic domain) into a process term.

A. Basic Syntax of CCS

Let \mathcal{A} be a countably infinite collection of names, and the set $\overline{\mathcal{A}} = \{a \mid a \in \mathcal{A}\}$ be the set of complementary names (or co-names for short). Let $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$ be a set of labels, and $Act = \mathcal{L} \cup \{\tau\}$ be the set of actions, where τ denote the activities which are not externally visible. Let \mathcal{K} be a countably infinite collection of process names.

The collection of CCS expressions is given by the following grammar:

$$P, Q ::= K \mid \alpha.P \mid \sum_{i \in I} P_i \mid P \mid Q \mid P[f] \mid$$

$P \setminus L$

where:

- K is a process name in \mathcal{K} ;
- α is an action in Act ;
- I is an index set;
- $f : Act \rightarrow Act$ is a relabelling function satisfying the following constraints:
 - $f(\tau) = \tau$ and
 - $f(\overline{a}) = \overline{f(a)}$ for each label a ;
- L is a set of labels.

B. Operational Semantics of CCS

CCS is formalized using axiomatic and operational semantics. To formally capture the understanding of the semantics of the language CCS, the collection of inference rules are therefore introduced as follows (a transition $P \xrightarrow{\alpha} Q$ holds for CCS expressions P, Q if, and only if, it can be proven using these rules):

$$\begin{array}{ll} \text{Rule 1} & \alpha.P \xrightarrow{\alpha} P \\ \text{Rule 2} & \frac{P_j \xrightarrow{\alpha} P'_j}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'_j} \quad j \in I \\ \text{Rule 3} & \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \\ \text{Rule 4} & \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \\ \text{Rule 5} & \frac{P \xrightarrow{\alpha} P', Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\tau} P \mid Q'} \end{array}$$

For a detailed introduction to the syntax and operational semantics of CCS, readers are referred to [17, 18].

C. Defining Valuation Functions

The valuation functions of IMWSC, and their corresponding semantic domains, semantic ranges are given in Tab. 1 (symbols $IMWSC$ denotes an IMWSC instance; P denotes process term in CCS; A denotes the set of atomic processes; $Activity$ denotes a set of activities; Act denotes the set of actions in CCS).

TABLE I

Valuation Functions and their Domains and Ranges

Valuation Function	Semantic Domain	Semantic Range
f_m	$IMWSC$	P
f_c	$Proc$	P
f_a	$Proc$	P
f_r	A	P
f_e	$Activity$	Act

where:

- $f_m(proc_r) = f_c(proc_1) \mid f_c(proc_2) \mid \dots \mid f_c(proc_n)$, where $proc_r, proc_i \in Service (1 \leq i \leq n)$, and $f_s(proc_r) = \emptyset \wedge f_s(proc_i) = proc_r$;
- $f_c(proc_i) = f_a(proc_i)$ iff $f_{pT}(proc_i) = a$;
- $f_c(proc_i) = f_r(proc_i)$ iff $f_{pT}(proc_i) = c$;
- $f_a(proc_i) = f_e(a_1) \cdot f_e(a_2) \cdot \dots \cdot f_e(a_n)$, where $a_i \in Activity \wedge f_e(a_i) = proc_i$, and $a_1 <_c a_2 <_c \dots <_c a_n$;
- $f_e(a_i) = !a_i$ iff $f_{aT}(a_i) = ii \vee f_{aT}(a_i) = ei$;
- $f_e(a_i) = ?a_i$ iff $f_{aT}(a_i) = io \vee f_{aT}(a_i) = eo$;
- $f_r(proc_i) = f_c(proc_1) \mid f_c(proc_2)$ iff $f_{pU}(proc_1) = proc_i \wedge f_{pU}(proc_2) = proc_i \wedge proc_i = proc_1 \parallel proc_2$;
- $f_r(proc_i) = f_c(proc_1) \cdot f_c(proc_2)$ iff $f_{pU}(proc_1) = proc_i \wedge f_{pU}(proc_2) = proc_i \wedge proc_i = proc_1 \prec proc_2$.

By means of the valuation functions defined in Tab. 1, an algorithm aiming at translating an IMWSC instance to CCS terms can be developed:

Algorithm. IMWSC_Instance_to_CCS

INPUT: IMWSC Instance

OUTPUT: The corresponding CCS terms

Process $Trans_{f_m}(IMWSC \text{ Instance})$

- {
1. $Str \text{ Exp} = \text{Empty}$;
 2. For each p in $Proc$;
 3. $\text{Exp} = \text{Exp} \mid Trans_{f_c}(p)$;
 4. Return Exp ;

```

}
Process Trans_fc ( process p ∈ Proc )
{
1. If ( process Type = basic )
   { Return Trans_fa ( p ) };
2. Else { Return Trans_fr ( p ) };
}
Process Trans_fa ( process p_i ∈ Proc )
{
1. Str name = getName( p ) ;
2. SET name = NIL ;
3. For each activity a in Activity_i of process p_i
4. If ( activityType = output )
   name = ! getName( a_i ). name ;
5. Else If ( activityType = input )
   name = ? getName( a_i ). name.
6. RETURN name ;
}
Process Trans_fr ( process p_i ∈ Proc )
{
1. Str Exp = Empty ;
2. For each subService u_j of process p_i
3. If ( compositionType of p_i is parallel )
   Exp = Exp | Trans_fc ( u_j ) ;
4. Else Exp = Exp. Trans_fc ( u_j ) ;
5. Return Exp ;
}

```

If the IMWSC instance to be translated comprises m basic processes, and $n = \max \{ num(p_i) \}$, where $num(p_i)$ returns the number of the activities contained in process p_i , the complexity of above algorithm will be $O(m \times n)$.

V. Case Study: Application of IMWSC to a Concrete Scenario

A. Abstracting the Interaction of Web Services

We will investigate the application of IMWSC in a simple scenario. There are three services involved in this scenario:

- The *Client Service*, which need to find out some useful information (for convenience, client here is considered as a service);
- The *Response Service*, which is responsible for dealing with information inquiry requests;
- The *Information Service*, which acts as a database and providing the useful information.

The business process of this scenario is introduced briefly as follows:

1. The *Response Service* receives a request from the *Client Service* which need to find out some useful information;
2. The *Response Service* contacts the *Information Service* and relay the information inquiry request;
3. The *Response Service* answers the questions to the *Client Service*.

Fig. 3 presents an illustration of the structure of this scenario, where

- A service is visualized by a rectangle (with round angles);
- A state of a service is visualized by a circle (the initial and the terminative states of a service are visualized by icons \odot , \ominus respectively);
- A transition between states is visualized by an arrow (with curve line), from the source state to the target state ;
- The supply channels of services in this scenario is visualized by a pair of parallel arrows (with opposite directions).

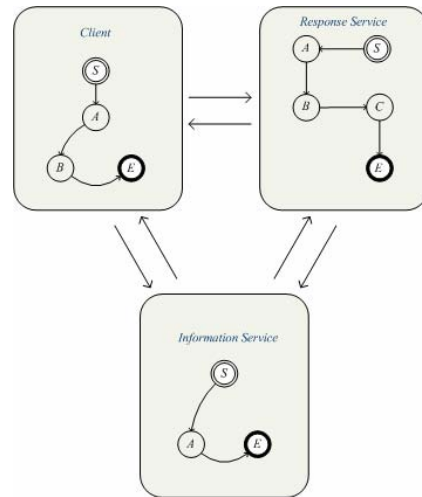


Figure 2. A Scenario of Interaction of Services

By applying IMWSC, the interaction process of services in this scenario is described as follows:

$$f_{con}(\text{Client}) = \{ cReq, cAsk, cInquiry, cInfo \};$$

$$f_{con}(\text{Reponse}) = \{ rReq, rAsk, rInquiry, rAnswer \};$$

$$f_{con}(\text{InfoS}) = \{ iAnswer, iInfo \}.$$

$$f_{aT}(cReq) = ii; f_{aT}(cAsk) = io; f_{aT}(cInquiry) = ii;$$

$$f_{aT}(cInfo) = io; f_{aT}(rReq) = io; f_{aT}(rAsk) = ii;$$

$$f_{aT}(rInquiry) = io; f_{aT}(rAnswer) = ii;$$

$$f_{aT}(iReq) = io; f_{eT}(iInfo) = ii; f_{aT}(iAnswer) = io.$$

$$cReq <_c cAsk <_c cInquiry <_c cInfo;$$

rReq <_c rAsk <_c rInquiry <_c rAnswer;
iAnswer <_c iInfo.

< rReq, cReq > ∈ R_a; < cAsk, rAsk > ∈ R_a;
< rInquiry, cInquiry > ∈ R_a; < cInfo, iInfo > ∈ R_a;
< iAnswer, rAnswer > ∈ R_a;

By means of the semantics of IMWSC defined in Section 4, the corresponding CCS terms translated are as follows:

Client = ! Req. ? Ask. ! Inquiry. ? Info. nil ;

Response = ? Req. ! Ask. ? Inquiry. ! Answer. nil;

InfoS = ? Answer. ! Info. nil;

Scenario = (Client | Response | InfoS) / { req, ask, info, Inquiry, Answer }

B. Verifying the Interaction of Web Services

CCS is an effective modeling language which has available supporting tool CWB-NC (Concurrency Workbench of the New Century, CWB-NC for short)^[20]. We use this tool to reason on and verify the behavior of an instance of IMWSC.

Using the supporting tool of CCS, i.e., CWB-NC, aims at assist the design and verification of a system. Applying CCS in the design phase of a system is helpful to show explicitly the interaction of the components that compose this system; after the model of a system has been constructed, modal μ -calculus^[23] can be used to reason on the system behavior. For a detailed introduction to modal logic, readers are referred to, for example, [19, 23].

One type of verification supported by the tool is reachability analysis. Here, as in each type of verification, our first step in using the tool is to write a description of the system supported by CWB-NC. The description is then parsed by the tool and checked for syntactic correctness. We then give a logical formula describing a “bad state” that the system should never reach. Given such a formula and system description, CWB-NC explores every possible state the system may reach during execution sequence and checks to see if a bad state is reachable. If a bad state is detected, a description of the execution sequence leading to the state is reported to the user. Many bugs such as deadlock and critical section violation may be found using this approach^[20].

Correct termination is one of the main properties a proper web service should satisfy. We use *can_terminate* to define the state of termination of a system. And the explanation for this state is as follows:

can_terminate is true of a system if it will reach a terminative state. We express this property the system should have in modal μ -calculus:

prop *can_terminate* =
min X = [-]ff ∨ <->X

Reachability analysis is actually a special case of a more general type of verification called model checking. In the model checking approach a system is again described using a design language and a property the system should have is formulated as a logical formula^[20].

Another type of verification supported by CWB-NC involves using a design language for defining both systems and specifications. Here the specification describes a system behavior more abstractly than the system description^[20]. A relation, i.e., *Observational equivalence* needs to be introduced before we conduct this type of verification.

Observational equivalence is useful in verification as they lay the conceptual basis for deciding that the behavior of two web services can be considered to be the same. They can also be used as a tool for reducing verification effort by replacing a process by a smaller (in size), but equivalent one. The bisimulation equivalence between two processes is a relation between their evolutions such that for each evolution of one of the services there is a corresponding evolution of the other service such that the evolutions are observationally equivalent and lead to processes which are again bisimilar. This characterization of the behavior of web services using the notion of bisimulation helps service designer optimize composite services by, e.g., changing their component web services with equivalent ones. Another motivation is customization of services. To enhance competitiveness a service providers may modify their service for customers' convenience and this customized service must conform to the original one. Formally, the relation of observational equivalence is defined as:

Definition 1 [Weak Transitions]^[23]:

- $q \xRightarrow{\varepsilon} q'$ iff $q = q_0 \xrightarrow{\tau} q_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} q_n = q'$, $n \geq 0$;
- $q \xrightarrow{\tau} q'$ iff $q \xRightarrow{\varepsilon} q'$;
- $q \xrightarrow{\alpha} q'$ iff $q \xRightarrow{\varepsilon} q_1 \xrightarrow{\alpha} q_2 \xRightarrow{\varepsilon} q'$, ($\alpha \neq \tau$).

Definition 2[Observational Equivalence]^[23]:

Let $S \subseteq Q \times Q$. The relation S is a weak bisimulation relation if whenever $q_1 S q_2$ then:

- $q_1 \xrightarrow{\alpha} q'_1$ implies $q_2 \xRightarrow{\alpha} q'_2$ for some q'_2 such that $q'_1 S q'_2$;
- $q_2 \xrightarrow{\alpha} q'_2$ implies $q_1 \xRightarrow{\alpha} q'_1$ for some q'_1 such that $q'_1 S q'_2$.

q_1 and q_2 are observationally equivalent, if $q_1 S q_2$ for some weak bisimulation relation S , written $q_1 \approx q_2$.

In this scenario, Client is considered as a service which interacts with the composition of the services *Response* and *InfoS*.

The behaviour of the composition of the services *Response* and *InfoS* can be described in two ways:

1. The system description of the composition of services *Response* and *InfoS* is:

$$Response = ? Req. ! Ask. ? Inquiry. ! Answer. nil;$$

$$InfoS = ? Answer. ! Info. nil;$$

$$Info_Response = (Response | InfoS) / \{answer\} ;$$

2. The specification of this composition is:

$$Spe = ? Req. ! Ask. ? Inquiry. ! Info. nil$$

Command 'eq -S obseq' of CWB-NC tool can be used to examine whether or not two processes are observationally equivalent. By executing this command, we know that processes *Info_Response* and *Spe* are observationally equivalent.

VI. Conclusions and Future Work

Formal description and verification of the interaction of web services is an important research field. After the description and verification of a practical application of web service, we come to a conclusion that IMWSC has very good capability in abstracting, simulating, and analyzing a scenario of the interaction process of web services, which will facilitate the correct implementation.

Currently many service composition methods do not take into account abstracting and analyzing the interactive features of services in a composition. Therefore it is apt to make mistakes when using these methods. Our work is an attempt to abstract and verify the interaction process of web services which will make the composition process more reliable.

Further work will involve defining the way IMWSC instances are composed. An instance of IMWSC model defined only one scenario of the interaction process of web services. To model the complete interaction process of web services, there is a need for composing the instances of IMWSC model. Since Petri nets are a well known formal model that is capable of defining the composition process, we plan to compose the instances by using Petri net. In our further work, we will present the fixed point property of IMWSC model. The fixed point property indicated the outcome of each instance of IMWSC model is determinate, in other words, each of the module has only one terminative state. This property lays the mathematical foundation for mapping a module to a transition in a Petri net.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China under Grant No.60573087.

REFERENCES

- [1] Rachid H., Boualem B.. A Petri net based model for Web service composition. Proc. of the 14th Australian Database Conference on Database Technologies, Adelaide, South Australia, 2003: 191-200.
- [2] Salaün G., Bordeaux L.. Describing and reasoning on Web services using process algebra. Proc. of the 2nd IEEE International Conference on Web Services, San Diego, California, USA, 2004: 43-51.
- [3] Schahram Dustdar, Wolfgang Schreiner. A survey on web services composition. Int. J. Web and Grid Services, 1(1): 1-30, 2005.
- [4] Muhammad Adeel Talib. Modeling the Flow in Dynamic Web Services Composition. Information Technology Journal, 3 (2): 184-187, 2004.
- [5] Casati F, Sayal M, and Shan M C. Developing e-services for composing e-services. Proc. of the 13th International Conference on Advanced Information Systems Engineering (CAiSE2001), Interlaken, Switzerland, 2001: 171-186.
- [6] Patil A, Oundhakar S, Sheth A, Verma K. METEOR-S Web service annotation framework [A]. Proc. of the 13th International World Wide Web Conference (WWW2004)[C], New York, USA, May 2004: 553 – 562.
- [7] Benatallah B, Sheng Q Z, Dumas M. The Self-Serv Environment for Web Services Composition[J], IEEE Internet Computing, 2003, 7(1): 40 – 48.
- [8] Medjahed B, Bouguettaya A, Elmagarmid A K. Composing Web services on the Semantic Web[J], VLDB Journal, 2003, 12(4): 333 – 351.
- [9] Shankar R P, Armando F. SWORD: A developer toolkit for Web service composition. Proc. of the 11th International World Wide Web Conference, Honolulu, Hawaii, USA, May 2002, 786 – 810
- [10] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN Planning for Web Service Composition using SHOP. Journal of Web Semantics, 1(4):377-396, 2004.
- [11] Zhang Jia , Chung Jen Yao , Chang C. K. . WS-Net: A petri-net based specification model for web services. Proc. of the 2nd IEEE International Conference on Web Services, San Diego, California, USA, 2004:420-427.
- [12] Tao X. F. Formalizing Web service and modeling Web service based on object oriented Petri net. Lecture Notes in Computer Science. Berlin, Heidelberg. Springer-Verlag, 2004.
- [13] A. Ferrara. Web Services: a Process Algebra Approach”, Proc. of the 2nd Int. Conf. on Service-Oriented Computing (ICSOC'04), ACM Press, 2004: 242-251.
- [14] Li Bao, Weishi Zhang, Xiuguo Zhang. Describing and Verifying Web Service Using CCS. In Proc. of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies, 2006: 421-426.
- [15] R. Milner. Communication and Concurrency. Prentice Hall, 1989.
- [16] R. Milner. A calculi of Communication and Concurrency. Spinger-Verlag, 1980.
- [17] Colin Fidge. A comparative introduction to CSP, CCS and LOTOS. Technical Report, Department of Computer Science, University of Queensland, Brisbane, Australia, April 1994. Available at: <http://sky.fit.qut.edu.au/~fidgec/Publications/fidge94g.pdf>.
- [18] Luca Aceto, Kim G. An Introduction to Milner's CCS. Larsen. Technical Report, Department of Computer Science, Aalborg University, Aalborg, Denmark, March 2005. Available at:

<http://www.dimi.uniud.it/miculan/Didattica/MFGC04/intro2ccs.pdf>.

- [19] C. Stirling. Modal logics for communicating systems. *Theoretical Computer Science*, 49(2-3): 311-347, 1987.
- [20] R. Cleaveland and S. Sims. The NCSU concurrency workbench. In R. Alur and T. Henzinger, editors, *Proceedings of the 8th International Conference on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, NJ, USA, 1996: 394-397.
- [21] K. Schmidt and C. Stahl. A Petri net semantic for BPEL4WS - validation and application. *Proc. of the 11th Workshop on Algorithms and Tools for Petri Nets*, Paderborn, Germany, 2004: 1-6.
- [22] B. Schlingo, A. Martens, and K. Schmidt. Modeling and model checking web services. *Proc. of the 2nd*

International Workshop on Logic and Communication in Multi-Agent Systems. volume 126 of *Electronic Notes in Theoretical Computer Science*, Nancy, France, August 2004: 3-26.

- [23] M. Hennessy and R. Milner, Algebraic laws for nondeterminism and concurrency, *J. Assoc. Comput. Mach.*, 32: 137-161, 1985.
- [24] Hull R., Su J. (2005) Tools for Composite Web Services: A Short Overview. *SIGMOD Record*, 34 (2): 86-95, 2005.



Li Bao received the BS degree in computer science from Dalian Nationality University, China, in 2003, and the MS degree in computer science from Dalian Maritime University, China, in 1996. From 2006 to date, he works as a PhD candidate in the Institute of Software Engineering, Dalian Maritime

University, China. His research interests include distributed computing, software engineering, and formal description techniques.



Weishi Zhang received the BS degree in computer science from Xi'an Jiaotong University, China, in 1984, and the MS degree in computer science from the Chinese Academy of Science, China, in 1986. He received the PhD degree in computer science from the University of Munich, Germany, in

1996. From 1986 to 1990, he was an assistant researcher at the Shenyang Institute of Computing, Chinese Academy of Science, China. From 1990 to 1992, he was a visiting scholar at Passau University, Germany. From 1992 to 1997, he was an assistant professor at the University of Munich, Germany. In 1997, he joined the Department of Computer Science, Dalian Maritime University, China, where he is currently a professor of computer science. His research interests include distributed computing, software engineering, software architecture, formal specification techniques, and program semantics models.



Xiong Xie works as a PhD candidate in the Institute of Software Engineering, Dalian Maritime University, China. Her research interests include distributed computing, software engineering, and formal description techniques