An Interface Theory Based Approach to Verification of Web Services [*]

Zhenbang Chen, Ji Wang, Wei Dong, Zhichang Qi
National Laboratory for Parallel and Distributed Processing, Changsha, China
{*z.b.chen, jiwang, dong.wei*}*@mail.edu.cn*
W.L. Yeung
Department of Computing and Decision Sciences, Lingnan University, Hong Kong, China
wlyeung@ln.edu.hk

Abstract

The verification of web services becomes a challenge in software verification. This paper presents a framework for verification of web service interfaces at various abstraction levels. Its foundation is the interface theory for web services, in which transaction features are incorporated. Within the framework, one may check non mutual invocation, compatibility and refinement of web services at signature, conversation and protocol levels. At protocol level, we present a model checking approach to verifying the protocol properties in Action Set Computation Tree Logic(ASCTL). The paper also discusses the integration of our framework into the web service development.

1. Introduction

Web service is emerging as a standard framework for service-oriented computing. It becomes a new challenge to ensure the high confidence of web service systems. As a formal foundation of component-based design, [1] proposed a theory of interface automata for specifying the interfaces of components. [2] presented a web service interface description language, which can describe the interfaces at three levels, i.e. signature, consistency and protocol. However, the transaction features have not been considered in the existing interface theories, while they are essential features for distributed computing, such as web service systems. We have extended the formalism of web service interfaces proposed in [2] to describe transaction information in all three levels of signature, conversation and protocol. In this paper, we present a multi-level verification framework to verify web service interfaces. In the framework, the interface behaviour of web service interfaces will be specified and verified with respect to the expected properties at different abstract levels. The remainder of this paper is organized as follows. Section 2 briefly presents the underlying interface theory, and Section 3 proposes the verification framework and show the details in verifying protocol property. Section 4 discusses the integration of our framework into the web service development. Section 5 discusses some related works and concludes the paper.

[*]Supported by the National Natural Science Foundation of China under Grant No.60233020, 60303013, 90612009; the National High-Tech Research and Development Plan of China under Grant No.2005AA113130; the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321802; Program for New Century Excellent Talents in University under Grant No.NCET-04-0996.

2. Web Service Interface Theory

A web service interface description contains some method declarations, and clients can use the functionalities of web services through method calls. A web service may provide or request some methods which may return some different values. An *action* is an instance of a method call. In the perspective of action, web service interface behaviour contains *three parts*. The *first part* is the normal behaviour of action invocations. If an exception action is invoked, after which the corresponding fault handling behaviour in the description should be taken, which is the *second part*. If an exception action can invoke some successful actions before the exception occurrence, the successful actions should be compensated by the corresponding compensation behaviour which is the *third part* of the interface behaviour. Figure 1 shows a supply chain management system with transactions. *SellItem* is the method provided by web service **Shop**, and $\langle SellItem, FAIL\rangle$ is one of its provided actions.

There are different detailed interface descriptions from web service providers. For this reason, we proposed the interface theory for describing the transaction information at three different abstract levels of signature, conversation and protocol, which is same as [2]. As a shorthand, brief description of the interface theory will be given as follows.

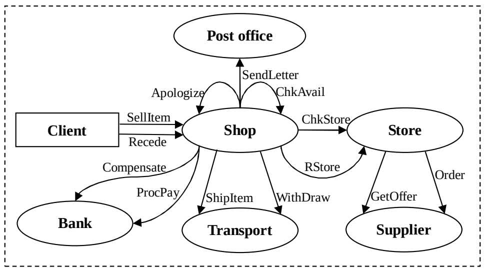

Figure 1. Supply chain management system.

According to the three parts of interface behaviour, a web service *signature interface* has three partial functions: $\mathcal{S}$, $\mathcal{S}_{\mathcal{C}}$ and $\mathcal{S}_{\mathcal{F}}$. The definitions of all functions are same as $\mathcal{A} \rightarrow 2^{\mathcal{A}}$, where $\mathcal{A}$ is a set of actions. $\mathcal{S}$ assigns to an action a a set of actions which a can normally invoke. $\mathcal{S}_{\mathcal{C}}$ assigns to an action a a set of actions that can be invoked by the compensation for a. $\mathcal{S}_{\mathcal{F}}$ assigns to an action a a set of actions that can be invoked by the fault handling for a.

An action may invoke different action sets in different cases, which is the reason for proposing *conversation interface*. A conversation is a set of actions that are invoked together. A conversation interface also has three partial functions: $\mathcal{E}$, $\mathcal{E}_{\mathcal{C}}$ and $\mathcal{E}_{\mathcal{F}}$. The definitions of all functions are same as $\mathcal{A} \rightarrow \omega(\mathcal{A})$. $\omega(\mathcal{A})$ is the set of expressions over the action set $\mathcal{A}$ using the binary operators $\sqcap$ and $\sqcup$, and the constant $\top$. The meanings of functions are similar to those of signature interface, except that each function assigns to an action a conversation expression to describe the interface behaviour.

A conversation is a set of actions, which does not have any sequence information. For indicating the sequences of action invocations, we propose *extended protocol automaton*, which is a triple $(\mathcal{A}, \mathcal{L}, \delta)$, where $\mathcal{A}$ is a set of actions, $\mathcal{L}$ is a set of locations, and there are two special locations $\bot, \boxtimes$ in $\mathcal{L}$, $\bot$ is the return location, and $\boxtimes$ is the exception location, and $\delta \subseteq (\mathcal{L} \setminus \{\bot, \boxtimes\}) \times Terms(\mathcal{A}) \times \mathcal{L}$ is the transition relation set. $Terms(\mathcal{A})$ is the term set whose elements indicate different modes of method invocations, such as thread creation, choice, and parallel executions. A *protocol interface* has an extended protocol automaton G and three partial functions: $\mathcal{R}$, $\mathcal{R}_{\mathcal{C}}$ and $\mathcal{R}_{\mathcal{F}}$, whose meanings are similar to those in signature interface, except that each function assigns to an action the start location in the extended protocol automaton. The definitions of all functions are same as $\mathcal{A} \rightarrow \mathcal{L}$.

Signature interface and conversation interface describe the static invocation relations of web service interfaces. Protocol interface describes dynamic web service interface behaviour. Given a protocol interface $\mathcal{T}$ and an action $a \in dom(\mathcal{R})$, the interface behaviour invoked by a can be transformed into a *labeled transition system* (LTS).

Compared with the interface theory in [2], we extend it with transaction description mechanism at all three levels. In each level, we add compensation function and fault handling function, through which the semantics of the interface can incorporate features of long running transaction.

3. Verification Framework

For ensuring the high confidence of web service systems, it is desired to verify web service interfaces with respect to the expected properties. The presented web service interface theory provides the foundation for verification. According to three abstract levels, we propose a verification framework for verifying web service interfaces. The sketch of the framework are shown in Figure 2. The verification can be taken on signature, conversation and protocol levels, and non mutual invocation, compatibility and substitutivity can be checked in each level. The compatibility and substitutivity between web service interfaces will be checked with respect to a set of conditions derived from the interface theory. In conversation and protocol levels, part of the expected properties will be application-related, and will be specified and verified specially.

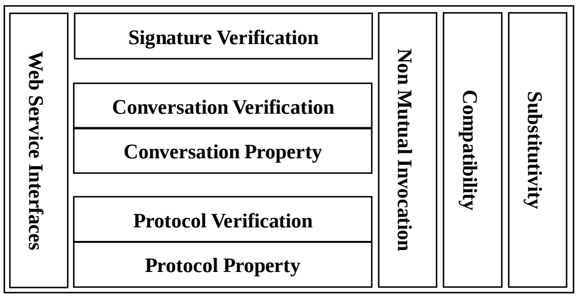

Figure 2. Sketch of verification framework.

For a web service system, if two actions can invoke each other mutually, the system resource may be exhausted eventually. Therefore it is desired that the mutual invocation should be forbidden in web service systems. Because signature interface describes direct invocation relation between web service interfaces, the non mutual invocation property can be checked on it. A signature interface is non mutual invocation if any two supported actions can not invoke each other mutually. For conversation and protocol interface, we can calculate their underlying signature interfaces, on which we can check the non mutual invocation property.

Conversation interface can describe different cases for the same action invocation. Some non-temporal properties can be verified on it. The property can be specified by *conversation property* ($\mathcal{CP}$), whose form is defined as follows, where $a \in \mathcal{A}, \mathcal{D} \subseteq \mathcal{A}$ and $a \notin \mathcal{D}$.

$$\mathcal{CP} \ :: \ a \rightarrow \diamond\mathcal{D} \mid a \rightarrow \Box\mathcal{D} \mid a \nrightarrow \mathcal{D} \mid a \nrightarrow \forall\mathcal{D} \mid a \nrightarrow \exists\mathcal{D}$$

The meanings of different property forms are given as follows: 1) $a \rightarrow \diamond\mathcal{D}$ specifies that action a can invoke a conversation which includes all actions in $\mathcal{D}$, 2) $a \rightarrow \Box\mathcal{D}$ specifies that every conversation invoked by action a includes all actions in $\mathcal{D}$, 3) $a \nrightarrow \mathcal{D}$ specifies that every conversation invoked by action a does not include all actions in $\mathcal{D}$, 4) $a \nrightarrow \forall\mathcal{D}$ specifies that every conversation invoked by action a does not include any action in $\mathcal{D}$, 5) $a \nrightarrow \exists\mathcal{D}$ specifies that action a can invoke a conversation which does not include any action in $\mathcal{D}$.

Some temporal properties can be verified on protocol interface. The *protocol property* must be formed in $a \rightarrow \varphi$, where $a \in dom(\mathcal{R})$ and φ is the formula in *Action Set Computation Tree Logic* (ASCTL), whose definition is given as follows, where $D \subseteq \mathcal{A}$.

$$\begin{aligned} &\chi :: true \mid false \mid D \mid \neg\chi \mid \chi \wedge \chi' \\ &\varphi :: true \mid false \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \mathbf{E}\gamma \mid \mathbf{A}\gamma \\ &\gamma :: [\varphi\{\chi\}\ \mathbf{U}\ \{\chi'\}\varphi'] \mid [\varphi\{\chi\}\ \underline{\mathbf{U}}\ \{\chi'\}\varphi'] \end{aligned}$$

Compared to ACTL in [4], the syntax and semantics of ASCTL are similar except that the action set is used instead of the single action. So, for the semantics of ASCTL, the labels of transitions in LTS model are action sets, and we define $A \models D$ iff $A \cap D \neq \emptyset$, where A is the transition label and D is the action set in ASCTL formula. The derived ASCTL operators, such as **EF**, **AF**, **EG** and **AG**, are defined as in [4].

The non mutual invocation checking and conversation property verification are straightforward, and model checking method is used to verify protocol properties. The method for model checking is same as [4], which used symbolic model checking method based on fixed point calculation [6] for ACTL verification. The verification process of protocol property is shown in Figure 3.

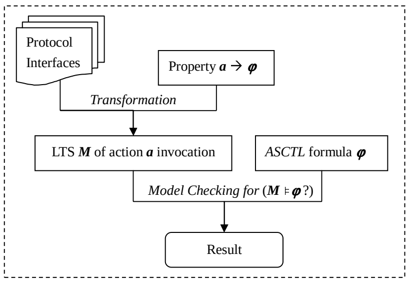
Protocol Interfaces
Property $a \rightarrow \varphi$
Transformation
LTS M of action a invocation
ASCTL formula φ
Model Checking for ($M \models \varphi$?)
Result

Figure 3. Protocol property verification.

First, the corresponding LTS of the property should be generated, and the LTS only contains the transitions whose labels are external action sets. Next, we can use the presented method to verify the ASCTL formula in the property.

Given two web service interfaces, we want to check whether they can cooperate properly. First, two services can

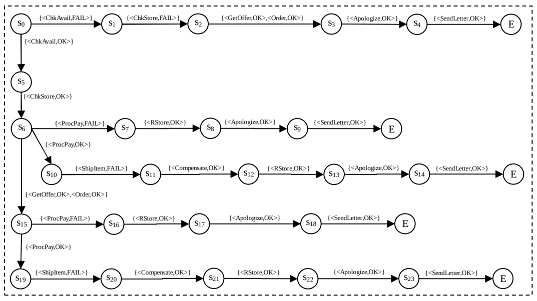

Figure 4. LTS of $\langle SellItem, FAIL\rangle$ invocation.

not support same actions. Second, the new service interface, which is composed by them, should be well-formed. To enable top-down design, it is desired to replace a web service in a system with a new web service without affecting the running of the system. After replacement, all parts of the system can still cooperate properly as before. The main idea of the substitutivity is the new web service should guarantee more and assume fewer than the old web service. Intuitively, for a web service, the defined methods of interface are the guarantees, and the remote calls are the assumptions.

Protocol Property	Result
⟨SellItem,FAIL⟩ → **AF** {⟨SendLetter,OK⟩}	True
⟨SellItem,FAIL⟩ → ¬ **E** [{¬ {⟨ProcPay,OK⟩}} **U** {{⟨Compensate,OK⟩}}]	True
⟨SellItem,FAIL⟩ → ¬ **E** [{¬ {⟨Compensate,OK⟩}} **U** {{⟨RStore,OK⟩}}]	False
⟨SellItem,FAIL⟩ → ¬ **E** [{¬ {⟨Compensate,OK⟩}} **U** {{⟨RStore,OK⟩}} **EF** {⟨Compensate,OK⟩}]	True

Table 1. The protocol property and the corresponding verification result.

After composing, the supply management system contains six web services. We can verify some properties of the composed system interface. For indicating the compensation and fault handling behaviour, the LTS of the $\langle SellItem, FAIL\rangle$ action invocation is shown in Figure 4, which is the projection to the external actions. The protocol properties for $\langle SellItem, FAIL\rangle$ and the corresponding results are shown in Table 1. The formula $\mathbf{AF}\{\langle SendLetter, OK\rangle\}$ represents that the shop must send an apologetic letter for the failure of purchasing. $\neg\mathbf{E}\ [\{\neg\{\langle ProcPay, OK\rangle\}\}\ \mathbf{U}\ \{\{\langle Compensate, OK\rangle\}\}]$ represents that the compensation for the payment must not occur before the payment. $\neg\mathbf{E}\ [\{\neg\{\langle Compensate, OK\rangle\}\}\ \mathbf{U}\ \{\{\langle RStore, OK\rangle\}\}]$ represents that it will never happen that if the compensation for the successful checking availability occurs, no compensation for the payment has occurred before. The last property represents that if the compensation for the payment occurs, it must not occur after the compensation for the successful checking availability. This property is verified to ensure the behaviour model satisfies the requirements of long running transaction.

4. Integration of the Verification Framework into Development Process

The verification framework can be employed in the process of web service development, to achieve a high assurance of the developed systems. Figure 5 shows an outline for incorporating verification activities in the development process.

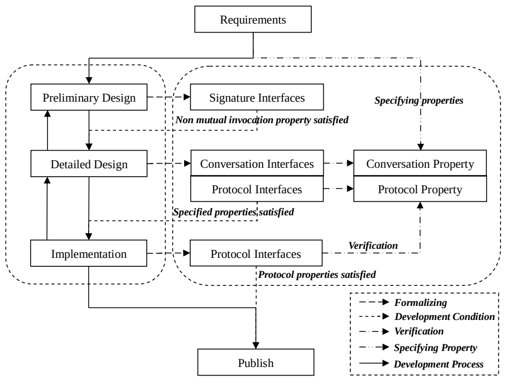

Figure 5. Integration of the verification framework into development process.

After acquiring requirements, the designer can specify some properties according to the requirements. When the designer finishes preliminary design, signature interface can be used for formalizing the interface description and non mutual invocation property can be checked. If the non mutual invocation property is satisfied, the designer can continue to do detailed design, otherwise some modification in preliminary design should be taken. When detail design is finished, some requirement properties specified before can be verified. If all requirements are satisfied, the designer can give the detailed design to the implementor, who may implement the actual web services by some composition languages, such as BPEL4WS[3]. After finishing implementation, the requirement properties should be verified on implementation model to ensure the correctness and consistency. If some requirement properties are not satisfied during these two steps, the designer or implementor should modify its design or implementation to ensure the requirement satisfied. Finally, the verified web service interfaces can be published to the web service registry and web services can be deployed to some service engines. The verification framework can also be used as the foundation for the registry to test the web services to be published.

5. Related Works and Conclusions

There are some researches on formalization and verification of web service interfaces [7, 8, 9]. Most of them have focused on formalization and verification using formalisms such as Petri-nets, state machine and process algebra. Their approaches are deficient in modeling and verifying the transaction behaviour of web service interfaces, especially in compensation and fault handling. Based on extending the formalism in [2], the approach presented in this paper can rigorously describe and verify the transaction behaviour of web service interfaces.

The paper presents a verification framework for the web service interfaces based on the interface theory. Different aspects of web service interfaces can be verified in the framework. In signature level, non mutual invocation can be checked. Conversation property can be verified on conversation level. Protocol property which is specified in ASCTL can be verified using model checking on protocol level. Besides that, compatibility and substitutivity can be checked on each level.

6. References

[1] L. de Alfaro, T. A. Henzinger, Interface automata, presented at the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering Vienna, Austria, 2001.

[2] D. Beyer, A. Chakrabarti, T.A. Henzinger, Web Service Interfaces, presented at 14th International World Wide Web Conference, Chiba, Japan, 2005.

[3] F. Curbera, Y. Goland, et al, Business Process Execution Language For Web Services, Version 1.0, 2002.

[4] R. Meolic, T. Kapus, Z. Brezocnik, Verification of concurrent systems using ACTL, presented at the IASTED International Conference on Applied Informatics AI'2000, Innsbruck, Austria, 2000.

[5] R.D. Nicola, A. Fantechi, S. Gnesi, and G. Ristori, An action based framework for verifying logical and behavioural properties of concurrent sytems, presented at 3th WorkShop on Computer Aided Verification, Aalborg, Denmark, 1991.

[6] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, and D.L. Dill. Symbolic Model Checking for Sequential Circuit Verification, IEEE Transactions On Computer-Aided Design of Intergrated Circuits and Systems, Vol. 13, No. 4, 1994.

[7] R. Hamadi, B. Benatallah, A Petri net-based model for web service composition, presented at 14th Australasian Database Conference, Adelaide, South Australia, 2003.

[8] H. Foster, S. Uchitel, J. Magee, J. Kammer, Compatibility for Web Service Choreography, presented at 3th IEEE International Conference on Web Services, San Diego, CA, 2004.

[9] X. Fu, T. Bultan, J. Su, Analysis of Interacting BPEL Web Services, presented at 13th International World Wide Web Conference, New York, USA, 2004