Predicting and Learning Executability of Composite Web Services

Masahiro Tanaka and Toru Ishida

Department of Social Informatics, Kyoto University
Kyoto 606-8501 Japan
mtanaka@ai.soc.i.kyoto-u.ac.jp, ishida@i.kyoto-u.ac.jp

Abstract. Configuring a composite Web service by setting endpoints reduces the cost of development, but raises the probability of a request message triggering runtime execution failures. Previous works on validation of composite Web services are not useful because the application developer cannot modify atomic/composite services and the specifications needed for validation are not always available. Therefore, in this paper, we address two issues: predicting the executability of composite Web services for each request message, and acquiring input specifications to improve the prediction. To resolve these issues, we model atomic/composite services in a formal specification. Moreover, we apply constraint acquisition algorithm to acquire input specifications of atomic Web services. We conduct an experiment in which the proposed method is applied to a composite Web service in practical use. The result shows that our method can detect almost all messages that will trigger execution failure at a rather early stage of specification acquisition.

1 Introduction

Various organizations have released Web services and standardized the interfaces of Web services. This makes it possible to develop composite Web services in the following way. The designer of a composite Web service provides his/her composite Web service in WS-BPEL or OWL-S through the combination of abstract atomic Web services, for which only the interfaces, and not the endpoints, are defined. We refer to such a composite Web service as an abstract composite Web service. The application developer simply sets endpoints for the abstract atomic Web services forming the abstract composite service; this identifies the concrete atomic Web services that will be actually invoked. We refer to such an implemented composite Web service as a concrete composite Web service.

However, a concrete composite Web service developed in the above way may suffer runtime failure. WSDL definitions for the abstract atomic Web service define only types of values of request messages, but do not define their valid range. Thus the execution of a concrete composite Web service which contains atomic Web services may fail for some request messages if it is configured by setting endpoints for the abstract atomic Web services. When a request message triggers execution failure of any atomic Web service in the composite Web service, the cost of executing all prior atomic Web services that is wasted.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 572–578, 2008.

Previous works have proposed methods with verification techniques such as petri net[1] and model checking[2,3] in order to prevent execution failure. However, these previous works are not useful because providers of concrete atomic Web services, designers of abstract composite Web services, and application developers reside in different organizations. Even if an application developer verifies a concrete composite Web and determines that it might suffer runtime failure, he/she can modify neither the concrete atomic Web services nor the abstract composite Web service. Moreover, the application developer cannot always perform the verification because the specifications of concrete Web services required for the verification are often unavailable.

Therefore, in this paper, we address the following issues:

- To ensure that a composite Web service is executable, we need to predict the executability of the composite Web service for each request message based on as much specifications as is known.
- To improve the accuracy of executability prediction, we need to acquire the specifications of concrete atomic Web services based on a success/failure of execution for each request message.

To predict executability, we model atomic and composite Web services in a formal specification. Moreover, we apply a constraint acquisition algorithm[4] to acquiring the input specifications of concrete atomic Web services.

2 Formal Specification to Model Web Services

To predict the executability of a composite Web service, we need specifications about input/output relation (Request message to response message mapping) and input specification (Request message validity) for each constituent atomic Web services.

To allow the above specifications to be checked, we model an atomic service as a module in the formal algebraic specification CafeOBJ[5], that consists of the following two operations.

`domain-`*service-name* This operation represents the input specification. This takes a request message to the Web service and returns true or false. True means executable and false means not executable.

`execute-`*service-name* This operation represents the input/output relation. This takes a request message to an atomic Web service and returns the response message of the atomic Web service.

We note, however, the input specification and input/output relations are not always completely known. Moreover, in general, it is impossible to describe the input/output relations completely. This is why we describe constraints on values or types of elements of request/response messages as far as are known.

Figure 1 shows the specifications of a machine translator Web service. First the definitions of data types and messages are imported (line 2). The response

```
 1: mod TRANSLATOR {
 2:   pr(LANGUAGE + TRANSLATOR-REQUEST + TRANSLATOR-RESPONSE)
 3:
 4:   op domain-translator : TranslatorRequest -> Bool
 5:   op execute-translator : TranslatorRequest
                                  -> TranslatorResponse
 6:
 7:   var e : TranslatorRequest
 8:
 9:   -- Source language must be English or Japanese
10:   eq domain-translator(e) =
11:       1*(e) == english or 1*(e) == japanese .
12:   -- Language of result is specified by target language
13:   eq get-language(execute-translator(e)) = 2*(e).
14: }
```

Fig. 1. Specification of a machine translator service

message named `TranslatorResponse` is a string which is the result of the translation into the target language. Next, the two operations which represent input specification and input/output specification are declared (lines 4-5). Finally, axioms are described as equations following `eq` (lines 10-11,13). In this example, the first axiom states that the first value of the request message (source language) must be English or Japanese. The second axiom states that the Web service translates a given string into the language that is specified by the second value of the request message (target language). "`n*`" is an operation on N-tuple which extracts the nth value.

Predicting the executability of a composite Web service requires the specifications of the composite Web service. Our approach is to create the specifications of a composite Web service by combining the specifications of its constituent atomic Web services.

In OWL-S or WS-BPEL, a composite Web service has nested structures. A control construct block contains atomic Web services or other control construct blocks. We follow this and recursively define the specifications of control construct blocks. To allow this, we consider a control construct block as a Web service and define it using the request/response message, the input specification, and the input/output relation. The block that contains all other blocks and atomic Web services corresponds to the composite Web service. We define two operations to represent the input specification and the input/output relation in the specification of each control construct block. Dataflows and constraints based on features of control constructs are represented as axioms in the specification.

3 Acquiring Input Specifications

Complete specifications of Web services for the prediction are not always known, especially in the case of Web services. Therefore, we propose a method that acquires the input specifications of atomic Web services to improve prediction accuracy.

In our model described in the previous section, input specifications of a Web service are represented as a logical formula. Thus we adopt the constraint

acquisition algorithm[4] to acquire the input specifications because the result of the acquisition can be represented as a logical formula in the formal specification.

In our method, a request message to an atomic Web service and the success/failure of the execution of the message are given to the constraint acquisition algorithm as a training example. The acquisition result can be easily transformed into descriptions in the formal specification by defining logical formulas in the formal specification that correspond to predefined predicates.

We explain below how to model the input specifications to apply the constraint acquisition algorithm. First we define a request message to a service which has k elements as $I = \{x_1, ..., x_k\}$. Next we define predicates which represent input constraints. For the sake of simplicity, we assume that the predicates have one or two variables. We refer to a unary constraint on an input value x_i to ith parameter as b_i. We also refer to a binary constraint on input values x_i, x_j to ith and jth parameters as $b_{(i,j)}$.

b_i and $b_{(i,j)}$ can be defined as $b_i : x_i \in \mathit{class}$ **or** $x_i \notin \mathit{class}$ and $b_{i,j} : \{x_i, x_j\} \in \mathit{class1} \times \mathit{class2}$ **or** $\{x_i, x_j\} \notin \mathit{class1} \times \mathit{class2}$ respectively. *class*, *class1* and *class2* represent any class. $x_i \in \mathit{class}$ indicates that x_i is an instance of *class*. $\mathit{class1} \times \mathit{class2}$ represents a Cartesian product set of *class1* and *class2*. Constraint library B_s contains b_i and $b_{(i,j)}$ for all known classes or pairs of classes.

The constraint acquisition algorithm works based on the above formalization. When a positive example is given, the constraint acquisition algorithm adds to formula K ($K = \mathit{true}$ in the initial state) the conjunction of negation of all constraints in the constraint library that the example does not satisfy. When a negative example given, it adds to K the disjunction of all constraints in the constraint library that the example does not satisfy. The acquisition result is the conjunction of literals that should be set to true in order to satisfy K.

In general, the set of literals that satisfy K is not unique. Thus we consider the possible sets of literals that satisfy K as a set of hypotheses, H, and define a partial order $\prec$ between hypotheses in H as follows: $h_i \prec h_j \equiv (\forall x \in X)[h_i(x) = \mathit{true} \rightarrow h_j(x) = \mathit{true}]$. X is a set of possible values of a request message. $h_i(x) = \mathit{true}$ means that the atomic Web service is executable for request message $x \in X$ under the hypothesis h_i. Our method performs prediction by reducing operation `domain-`*service-name* under all h_g defined as follows: $\{h_g \in H | h_g \not\prec (\forall h \in H)\}$. Our method cancels execution only if the results of reducing under all h_g are `false`. This is because our prediction of executability involves detecting request messages that would cause service execution to certainly fail.

4 Experiment

We conducted an experiment to show how much our method can improve the efficiency of executing a composite Web service. We applied our method to a composite Web service shown in Fig. 2, which is for translation used in Language Grid[6]

The details of the process of the composite Web service are as follows:

1. Split the string given as a request message into words using Morphological Analyzer (MA)

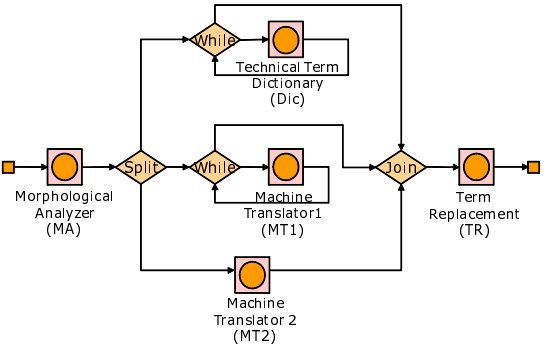

Fig. 2. Composite service for translation in a special domain

2. Concurrently execute the followings:
   (a) Translate all the words by Technical Term Dictionary (Dic)
   (b) Translate all the words by Machine Translator 1 (MT1)
   (c) Translate the string given as a request message by Machine Translator 2 (MT2)
3. Term Replacement (TR) replaces words in the string translated by MT2 with corresponding words translated by Dic

Suppose MA, MT1 and MT2 have the following input specifications.

- MA : Fail if any language other than Japanese or English is specified for the language of the given string.
- MT1, MT2 : Fail if the specified source language is different from the actual language of the given string or the given string is longer than 100 characters.

These input specifications lead to the possible failure of execution of the composite Web service as shown below.

- When the source language is neither Japanese nor English, MA fails and the cost of MA execution goes counted as waste.
- When the given string is longer than 100 characters, MT2 fails and the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic is counted as waste.
- When the given string has words in multiple languages (e.g. Japanese sentences often contain English words.), MT1 fails because the actual language of some of the given words differ from the source language specified. In this case, the cost of one execution of MA and MT2 and the iterated execution of MT1 and Dic are counted as waste as in the previous case.

We applied our method under the conditions described above. We assume that all input specifications of the atomic Web services are unknown in the initial state. Moreover, we defined predicates for the constraint acquisition algorithm. The predicates involve the input specifications of MA, MT1 and MT2. They

Fig. 3. Number of wasteful execution of atomic/composite services

represent classes for the source language, the target language, and the actual language of the given string (three classes for each) as described in Section 3. In this experiment, we also defined predicates to represent given string length (two classes: short and long). We generated request messages of all combinations of the classes for each element and executed the composite Web service by giving the messages in random order.

We counted failure of the execution of the composite Web service due to failure of any of the atomic Web services as one of the wasteful executions of the composite Web service. Similarly, we counted the sum of the executions of atomic Web services until one of the atomic Web service failed as the number of wasteful executions of atomic Web services. Figure 3(a)(b) compares the numbers of wasteful executions of the atomic/composite Web services shown in Fig. 2 before and after applying our method, respectively.

The figures show that the rate of increase in the number of wasteful executions saturates as the number of execution increases and more input specifications are acquired. In particular, Fig. 3(a) shows that our method works well in our example because it prevents some atomic Web services from being iteratively executed after the failure of some atomic Web service. This is very effective in reducing the cost of executing atomic Web services.

5 Conclusion

In this paper, we proposed a method for predicting the executability of composite Web services in order to reduce the cost of wasteful executions of atomic Web services. The major contributions of our method are as follows:

- We showed a model of Web services in a formal specification and applied it to predict the executability of a composite Web service for each request message by using a theorem prover.
- We applied the constraint acquisition algorithm in order to acquire input specifications of atomic Web services and showed that it improves the prediction of executability.

We conducted an experiment in which our method was applied to a composite Web service in practical use. The results showed that our method could detect almost all request messages that would cause execution failure. Compared to previous works, we assume that the application developer cannot modify concrete atomic Web services or abstract composite Web services because the stakeholders are in different organizations. This paper is the first work that focuses on the point and tries to reduce the wasteful execution of Web services by predicting the executability of each request message.

Acknowledgments

This works was partially supported by Grant-in-Aid for JSPS Fellows and Global COE Program "Informatics Education and Research Center for Knowledge-Circulating Society".

References

1. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: The 11th International Conference on World Wide Web (WWW 2002), pp. 77–88 (2002)
2. Ankolekar, A., Paolucci, M., Sycara, K.: Towards a formal verification of owl-s process models. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 37–51. Springer, Heidelberg (2005)
3. Fu, X., Bultan, T., Su, J.: Analysis of interacting bpel web services. In: The 13th conference on World Wide Web (WWW 2004), pp. 621–630 (2004)
4. Bessière, C., Coletta, R., O'Sullivan, B., Paulin, M.: Query-driven constraint acquisition. In: The 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 50–55 (2007)
5. Futatsugi, K., Nakagawa, A.: An overview of cafe specification environment-an algebraic approach for creating, verifying, and maintaining formal specifications over networks. In: The 1st International Conference on Formal Engineering Methods, pp. 170–181 (1997)
6. Ishida, T.: Language grid: An infrastructure for intercultural collaboration. In: IEEE/IPSJ Symposium on Applications and the Internet (SAINT 2006), pp. 96–100 (2006)