ISSN 1000-9825, CODEN RUXUEW E-mail: jos@iscas.ac.cn
Journal of Software, Vol.20, No.8, August 2009, pp.2102−2112 http://www.jos.org.cn
doi: 10.3724/SP.J.1001.2009.00568 Tel/Fax: +86-10-62562563
© by *Institute of Software*, *the Chinese Academy of Sciences*. All rights reserved.

基于BPEL的Web Service组合的数据流分析测试方法*

董文莉[1+], 胡建华[2]

[1](中国科学院 软件研究所,北京 100190)
[2](淮海工学院 现代教育技术中心,江苏 连云港 222005)

Test Method for BEPL-Based Web Service Composition Based on Data Flow Analysis

DONG Wen-Li[1+], HU Jian-Hua[2]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)
[2](Modern Education Technology Center, Huaihai Institute of Technology, Lianyungang 222005, China)
+ Corresponding author: E-mail: wenli@iscas.ac.cn

Dong WL, Hu JH. Test method for BEPL-based Web service composition based on data flow analysis. *Journal of Software*, 2009,20(8):2102−2112. http://www.jos.org.cn/1000-9825/568.htm

Abstract: As the Web Service composition becomes complex, testing to ensure their quality and reliability become crucial. This paper extends traditional data flow analysis to Web Service composition testing. A test method for BPEL-based Web Service composition based on data flow analysis is presented. The test method is based on a test model called WSCTM that captures data flow test artifacts of Web Service composition. With the considerations of the intra-activity, intra-service, and inter-service, testing for Web Service composition based on data flow analysis can be accomplished in three levels, and various flow graphs are used to describe the interaction within and between services in structure model. The def-use chains of the Web Service composition can be obtained based on above analysis method. As a result, test paths can be selected to satisfy given criteria in order to achieve a desired Web Service composition test coverage.

Key words: Web service; business process execution language for Web services; Web service composition; software testing; data flow analysis

摘 要: 随着 Web Service 组合变得越来越复杂,通过测试来保证服务质量和可靠性也变得越来越重要.将传统数据流分析方法扩展用于 Web Service 组合测试,提出了一种基于 BPEL 的 Web Service 组合的数据流分析测试方法.该方法基于一个测试模型:Web Service 组合测试模型 WSCTM,该测试模型可以捕获 Web Service 组合的数据流接口.采用基于服务的模型 WSCTM,数据流可以从 3 个视点来分析:服务间、服务内和服务实现构件间.从而,Web Service 组合的数据流测试可以在三层上得到实现.基于以上方法,可得到 Web Service 组合的定义-使用链,最终可产生满足既定测试标准以获得需求 Web 服务组合质量要求的测试路径.

关键词: Web 服务;业务流程执行语言;Web 服务组合;软件测试;数据流分析
中图法分类号: TP311 文献标识码: A

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z190, 2007AA010601 (国家高技术研究发展计划(863))

Received 2008-01-29; Accepted 2008-08-21

1 Introduction

With the extraordinary growth of distributed yet coordinated service-oriented frame work that can be directly or dynamically composed of Web Services through work-flow, the business process language for Web Services (BEPL, business process execution language for Web services)[1] is emerged for specifying and executing workflow specifications for Web Service composition invocation. Testing Web Service composition is considered a more challenging task than testing traditional program. Existing traditional software test methods have been found to be inadequate for Web Service composition testing[2−7]. The dynamical, heterogeneous, re-composite even during runtime, concurrent, and distributed nature makes Web Service composition difficult to understand and test. In addition, the technical complexity of Web Service composition, such as dynamical selection/re-composition, makes Web Service composition testing a more challenging task.

Recently, several tools and techniques of model checking over Web Services are discussed to verify the Web Service composition[8−14], aiming at guaranteeing discovering deadlock on services level etc. These studies employ different models, utilize different model checking approaches[8−11], or check different types of properties[12,13]. A common theme of existing approaches is that they treat atomic Web Service as black box. However, if the internal structure of Web Service is blank in the model specification, it is inherently hard to describe and check more delicate properties involving the effect and output of Web Service. It is insufficient to ensure the quality of Web Service composition without exercising its feasible paths without using structural test techniques (i.e., white-box testing). Therefore, to ensure correct data interactions and to provide sufficient test coverage, we extend one of the structural test techniques, data flowing testing, to Web Service composition testing.

Data flow testing is a commonly used structural test technique for guiding the selection of test paths based on the data flow information of programs[14−16]. It has been reported that test criteria based on a program’s data flow information can provide more adequate coverage than other structural test criteria, such as statement and branch criteria[17−19]. Moreover, empirical studies show that data flow testing is useful and promising in terms of cost saving and fault detection for complex program[20,21].

Existing data flow test techniques mainly study the data information of program variable and can be applied to individual functions and their interactions[15,16,18−20]. However, Web Service composition allows data to be stored in BPEL/WSDL document and services can be requested/provided dynamically so the role of provider and requestor is changed dynamically, and services can be re-composed during run-time. This introduces new data interactions. Furthermore, message correlation is introduced in BPEL mechanism, a BPEL defines correlation consisting of properties and enumerates the properties, and then references that set in activities. Thus, to test Web Service composition, data flow analysis needs to be extended to consider BPEL/WSDL documents and to cross the provider/requestor boundaries specified by traditional SOA model.

The rest of this paper is organized as follows. Section 2 presents a test model to represent the data flow test artifacts of Web Service composition. Section 3 proposes Web Service composition testing based on data flow analysis. Based on the method proposed in this paper, the analysis for an Automatic Transition Machine (ATM) application is presented in section 4 to illustrate our approach. Section 5 is the conclusion of the paper and possible future work on Web Service composition testing.

2 The Web Service Composition Test Model

Data flow analysis is concerned with the proper use of data in a program. In traditional program, data are stored in program variables. In Web Service composition, however, data can be stored not only in program (service implementation) variables but also in WSDL/BPEL documents transported from one service to another service or one operation to another operation. Therefore, traditional data flow techniques need to be extended to consider WSDL/BPEL documents that play an important role in Web Service composition.

In addition, the heterogeneous implementations of Web Service as well as the dynamic composition and re-composition make it difficult to locate and compute the data flow information of Web Service composition.

Furthermore, in traditional programs, the inter-procedural data flow analysis that is used to obtain the data flow information is based on the calling relations between the procedures. For Web Service composition, however, the data can be interacted in activities. Thus, to obtain the data interaction information between the Web Services, a more precise data flow analysis is needed.

WSDL/BPEL document is the basis of Web Service composition testing based on data flow analysis. WSDL is an XML encoded document describing networked services as a set of operations on messages specified by Schema[22]. Each step in the process defined by BPEL is called by a basic activity or structure activity. The WSCTM include various flow graphs so that different types of data flow test artifacts can be captured based on WSDL/BPEL. In particular, to represent the test artifacts, the WSCTM employs two models: the service model and the structure model. The service model crosses the provider/requestor boundaries, clearly describes Web Service composition in terms of service relationships based on the data interaction, and various flow graphs are used to describe the interaction within and between services in structure model.

2.1 The service model

The service model is used to describe Web Service composition in terms of service relationships based on the data interaction. In the service model, each entity of a Web Service composition is modeled as a service consisting of operations and variables. To facilitate the capturing of data flow information, the components of a service model are classified into three types: request implementation, Web Service finder, and activity (basic and structure). Each of them is defined as follows:

Definition 1. A request implementation is a script or program etc. that can send a request to a Web server.

Definition 2. A Web Service finder is a Web Service to provide the necessary access information of requested Web Service (e.g., location, name, function).

Definition 3. Activity is component that can be composed together to build a Web Service composition, that is, the basic activities and structure activities specified in BPEL.

To depict service and their interdependent relationships, a Service Relation Diagram (SRD) is employed. In particular, a $SRD=(V,E)$ is a directed graph, where V is a set of nodes representing the services, and $E\subseteq V*V$ is a set of edges representing the relationships between services. Moreover, to provide different levels of abstraction and detail, the SRD allows recursive definitions for service composition so that a complicated service composition can be represented hierarchically.

In addition, the relationships among services are classified into six types: inheritance, association, navigation, redirect, request, and response. The inheritance and association are similar to those of object-oriented program, to indicate the relationship between the parent service and its children service. The navigation, redirect, request, and response relationships are special types of association relationships that are introduced by service link or posed by Web Service composition based on BPEL. They are defined as follows.

Definition 4. $E_{Req}\subseteq V*V$ is the set of direct edges representing a request between a request implementation and a Web Service finder. For any two services $v_1,v_2\in V$, $(v_1,v_2)\in E_{Req}$ indicates that the Web Service finder v_2 is requested by the request implementation v_1.

Definition 5. $E_{Rs}\subseteq V*V$ is the set of directed edges representing a response between a request implementation and a Web Service finder. For any two services $v_1,v_2\in V$, $(v_1,v_2)\in E_{Rs}$ indicates that the Web Service finder v_1

responses the request of the request implementation v_2.

Definition 6. $E_N \subseteq V*V$ is the set of directed edges representing a navigation relation between two request implementations. For any two services $v_1,v_2 \in V$, $(v_1,v_2) \in E_{Rs}$ indicates that there is a navigation link from the request implementation v_1 to the request implementation v_2.

Definition 7. $E_{Rd} \subseteq V*V$ is the set of directed edges representing a redirect relation between two service finders. For any two services $v_1,v_2 \in V$, $(v_1,v_2) \in E_{Rs}$ indicates that the service finder v_1 redirect a request (via SOAP, HTTP, FTP etc.) to the service finder v_2.

Figure 1 shows the SRD for the ATM process described in Fig.4.

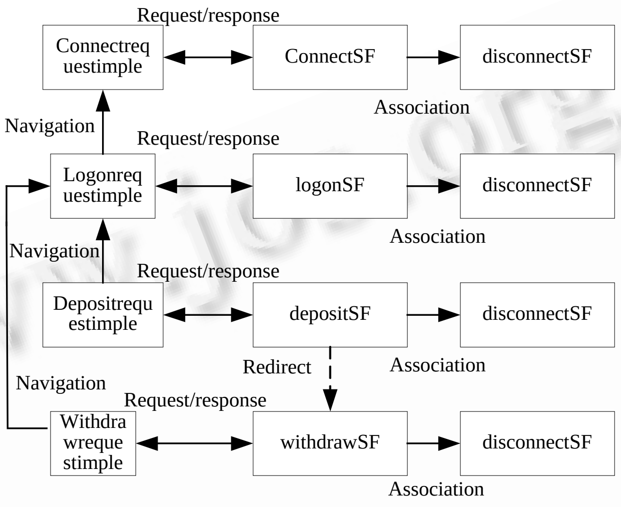

Fig.1 SRD for ATM process

A scenario, is that a user can deposit an amount of money by sending a request using request implementation *depositrequestimple*. The request implementation *depositrequestimple* logon the ATM transaction system server by navigating to the request implementation *logonrequestimple*, and the request implementation *logonrequestimple* connect network by navigating to the request implementation *connectimple*. The Web Service finder *connectSF* provides the relevant information of *connect* activity and ask the *connect* activity to accomplish the connect function and assign the message *connected* by sending a response to request implementation *Connectrequestimpl*e. Then, The Web Service finder *logonSF* provides the relevant information of *logon* activity and ask the *logon* activity to accomplish the logon function and assign the message *logonedon* by sending a response to request implementation *logonrequestimpl*e. If logon is successful, the Web Service finder *depositSF* provides the relevant information of service and the request implementation *depositrequestimple* invokes *deposit* activity and ask the *deposit* activity to accomplish the deposit function and assign the message operationRsp by sending a response to request implementation *depositrequestimple*.

In addition, to illustrate the relationship redirect defined in Definition 7, we can add an extra line in ATM process description as shown in Fig.4:

〈sequence name="redirectseq"〉 〈invoke partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="deposit" variable="operationReq"/〉 〈invoke partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="withdraw" variable="operationReq"/〉 〈/sequence〉

Figure 1 shows the case where a dashed arrow is used to indicate redirect relationship.

The service model can help testers understand the heterogeneous structures of Web Service composition. In addition, by modeling the entities of a Web Service composition as services, WSDL/BPEL documents can be integrated with the implementation so that the data flow information of the Web Service composition can be obtained in a consistent way. At the same time, the relationships among the services can also help testers identify the data interactions among interaction services and to compute the data flow information accurately.

2.2 The structure model

As mentioned in Section 2.1, the components of a service model are classified into three types: request implementation, Web Service finder, and activity. Correspondingly, the data interaction should be intra-activities, intra-services, and inter-service. The structure model is used to capture the data flow information of a Web Service composition. To capture the data flow information, three types of flow graphs are employed in the structure model. These flow graphs are described as follows:

Intra-Activity Control Flow Graph

The intra-activity control flow graph (ICFG) is used to describe the data flow information involving more than one operation. The intra-activity control flow arises from the operation calls while the intra-activity control flow is the process arrangement specified by BPEL basic activities. I.e., in ATM Web Service composition, for operation *logon*, the operation *connect* will be called, this will generate intra-activity control flow. The def-use chains for a variable that is defined in one operation and used in other operations can be obtained from ICFG.

Service Control Flow Graph

In Web Service composition, a user event may trigger a set of basic activities. We incorporate concurrent summarizing technique[23] into our analysis. Depending on the user's interactions, different event triggering sequences can result in various structure activities. Since the global variables can be defined or used by any operations within the services, different basic activities can result in different def-use chains for the variables of interest. Therefore, to ensure correct data interactions, it is necessary to compute the def-use chains associated with different basic activity invocation sequences.

To obtain the def-use chains as resulted from different structure activities, a service control flow graph (SCFG) is employed. The SCFG is constructed by connecting together all ICFGs of the operations within the service by structure activities. As shown in Fig.2, an extra node loop is connected to the nodes indicating operation *logon*, and *status* representing a selection. And an extra node loop is connected to the nodes indicating operation *deposit* and *withdraw* representing the alternative. This means after operation *connect*, which is executed once prior to other operation, then the process is connected to the loop node and direct the recycling execution of selective operation, *logon* and *status*, so does the node *logon*.

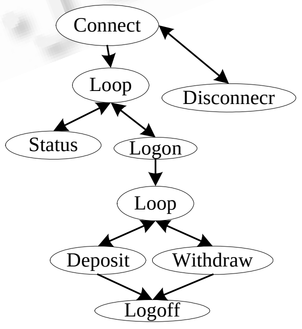

Fig.2 An example of SCFG

The SCFGs of different structure activities are shown in Fig.3[24]. The construction of the SCFG needs connecting the corresponding end node to the entry node satisfying pre-condition when p-use is marked on the edger.

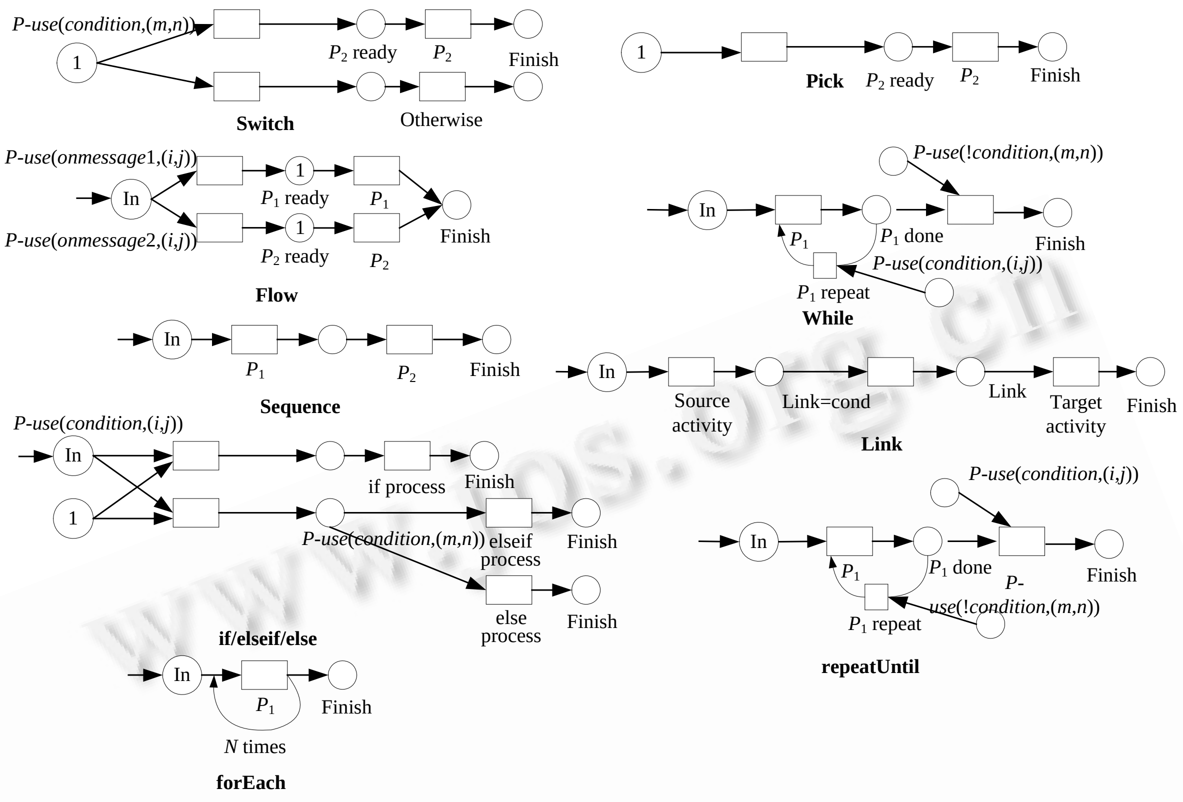

Fig.3 SCFG of different invoking sequences

Composite Control Flow Graph

One characteristic of Web Service composition is that an operation defined in a Web Service can be specified to an operation defined in another Web Service organized by BPEL. Thus, a message can be defined in one Web Service and used in another Web Service. However, in traditional program, no direct calling direction between the embedded functions across the program package and, hence, they cannot be used to capture this kind of data interaction. Therefore, to describe the data flow information between services, the composition control flow graph (CCFG) is introduced. Correlation Sets also are described in this graph to ensure the message correlation.

3 Testing for Web Service Composition Based on Data Flow Analysis

The test method is based on the data flow information extracted by the test model WSCTM. With the test method, Web Service composition testing can be accomplished in three levels.

3.1 Web service composition testing from service perspective

In the WSCTM, the entities of a Web Service composition are modeled as service, including request implementation, service finder, and activity. By using service to model the entities, we can derive data flow test cases from three different perspectives: intra-activity, intra-service, and inter-service.

From the intra-activity perspective, we consider the data flow information of individual operation within a basic activity as well as the data interaction among the operations through operation calls within the service. These can be obtained from the flow graph, ICFGs, of corresponding operations within a service. Thus, from the intra-activity data flow information, test cases can be derived to exercise the feasible paths in each basic activity of a Web Service composition.

Moreover, in Web Service composition, a service can obtain variables that are shared by multiple basic activities. These variables might have def-use chains that span over multiple basic activities. To ensure the shared data are manipulated correctly across different basic activities, data flow testing should take into account the "intra-service" data interactions. And from the relationship of services described in the service model, the intra-service data interactions can be based on structure activities. Their data flow information can be obtained from the flow graph, such as SCFCs and ICFGs, of corresponding activities.

The main focus of the inter-service is the data interactions between services. The data flow information can be obtained from the flow graphs, such as CCFGs, SCFGs, and ICFGs, of corresponding activities and services. Based on the inter-service data flow information, test cases can be derived for the service composition testing of Web Service composition.

3.2 Testing Web service composition based on data flow analysis

With the considerations of intra-activity, intra-service, and inter-service, testing Web Service composition based on data flow analysis can be accomplished in three levels. From the intra-activity perspective, the def-use chains need to be computed at the activity level. From the intra-service perspective, the def-use chains need to be computed at the service level. From the inter-service perspective, the def-use chains need to be computed at the service cluster level. Each of the three levels is described as follows:

Activity Level

Activity level testing is used to test a cluster of operations within a service for the message whose def-use chains involve more than one operation. An operation cluster is a set of operations that interact through operation calls within a service. With the calling relationships, the operation of a service can be grouped into a set of operation clusters. For an operation cluster, the activity level testing is required if the def-use chain for a definition in one operation consists of uses of the definition in other operations in the cluster. As described in Section 3, the def-use chains for an operation cluster can be computed from the ICFG of the cluster. From the def-use chains, test paths at the activity level then can be obtained.

Service Level

Service level testing is used to test the interactions of all the operations in a service. With different structure activity invocation sequence, the def-use chains of global variables within a service can be changed for the various composite constructs such as parallel (flow), choice (pick, switch), and sequence (sequence). As described in Section 3, the def-use chains for different structure activity invocation sequences can be computed from the SCFG of a service. Note that the number of structure activity invocation sequence can be infinite. As a result, a criterion is required in order to select a subset of the sequences to obtain the def-use chains.

Service Cluster Level

Service cluster level testing is used to test a cluster of services for the variables that have a def-use chain $\langle d,u\rangle$, where d is in one service and u is in another. Note that a service cluster is a set of services that are associated via message passing. For example, if service S_1 sends a message to service S_2, S_1 and S_2 are in the same service cluster. For message passing in service cluster level, such as data transmissions between two Web Services via SOAP protocol, test path can be obtained from the CCFG of related services as discussed in Section 3.

3.3 Discussion of the test approach

It should be pointed out that the major focus of our approach is to adapt traditional data flow analysis techniques to the context of Web Service composition for computing their def-use chains thoroughly. Especially, the approach takes into account the data flow information introduced by Web Service technologies, such as Web Service workflow (BPEL). As a result, the approach is able to compute important def-use chains that cannot be identified by applying conventional test methods to the scripts or programs in Web Service implementation. This approach also solves the insufficient problem resulting from lacking of the internal structure in the common theme of existing Web Service composition test approaches. It can describe and check more delicate properties involving the effect and output of Web Service composition.

By applying the approach to several examples, our experiences show that the def-use chains introduced by workflow, dynamic composition and re-composition of Web Service can be effectively identified. These def-use chains are critical in checking the data flow anomalies of Web Service composition. For example, in ATM process, by variable sessionID, both service ATM and ATMSupport are checked based on the data flow analysis. In particular, they are required to detect the structural errors caused by the misuses of variables in a Web Service composition implementation. In addition, a major part of the proposed approach can be automated. The SRD for a Web Service composition can be constructed automatically from the BEPL document and WSDL documents to achieve desired Web Service composition test coverage. Furthermore, in order to exercise the derived test paths, test data need to be generated. By combining with the automated test data generation methods, the approach can reduce the testing cost while improving the quality of a Web Service composition.

4 An ATM Transaction

To illustrate the test method for BEPL-based Web Service composition based on data flow analysis, let us consider the ATM service and ATMSupport service of the Web Service composition described by ATM BPEL presented in Fig.4.

Table 1 lists all the variables, their appropriate data flow test levels, and the def-use chains for the Web Service ATM and ATMSupport of an ATM transaction. Due to space limitation, the first line of BPEL document includes the variable definition is omitted in Fig.4, and the variables which are defined only in first line will be specified to be that in line 0. Based on our approach, all def-use chain can be thoroughly identified to serve as a reference for test case generation. These systematically identified def-use chains, when combined with appropriate test criteria and sound test strategies, can prove a cost-effective approach in uncovering Web Service composition defects. Taking variable *sessionID* as example to illustrate def-use chains, *sessionID* is defined in line 0, 2 (in service ATMSupport), 5 (in service ATM), 9 (initiated), and is used in line 4 (variable assigned in service ATM), 7 (operational variable), 20 (operational variable), 22 (variable used in service ATM), 34 (operational variable). So, 〈0,2〉 (from service ATM to ATMSupport) and 〈2,5〉 (from service ATMSupport to ATM) are Service cluster def-use chains; 〈5,9〉 (in service ATM), 〈9,22〉 (in service ATM), and 〈2,4〉 (in service ATMSupport) are service def-use chains; 〈5,7〉 (sessionID is operational variable in line 7), 〈9,20〉 (sessionID is operational variable in line 20), 〈9,34〉 (sessionID is operational variable in line 34) are activity (operation cluster) def-use chains.

```
0 …
1 ⟨receive partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="connect"
   variable="connectReq" reateInstance="yes" name="ATMConnectReceive"/⟩
2 ⟨invoke partnerLink="atmSupport" portType="support:atmSupportServicePT"
   operation="nextSessionID" inputVariable="newSessionReq" outputVariable="newSessionRsp"/⟩
3 ⟨assign⟩ ⟨copy⟩
4 ⟨from variable="newSessionRsp" part="sessionID"/⟩
5 ⟨to variable="sessionMsg" part="sessionID"/⟩
6 ⟨/copy⟩ ⟨/assign⟩
7 ⟨reply partnerLink="atmFrontEnd" portType="tns:atmServicePT" opera-tion="connect"
   variable="sessionMsg" name="ATMConnectReply"⟩
8 ⟨correlations⟩
9 ⟨correlation set="atmInteraction" initiate="yes"/⟩
10 ⟨/correlations⟩ ⟨/reply⟩ ⟨assign⟩ ⟨copy⟩
11 ⟨from expression="true()"/⟩
12 ⟨to variable="connected"/⟩
13 ⟨/copy⟩ ⟨/assign⟩
14 ⟨while condition="bpws:getVariableData('connected')"⟩
15 ⟨scope variableAccessSerializable="no"⟩
16 ⟨correlationSets⟩
17 ⟨correlationSet name="customerInteraction" properties="tns:customerid"/⟩
18 ⟨/correlationSets⟩
19 ⟨pick createInstance="no"⟩
20 ⟨onMessage partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="logon" variable="logonReq"⟩
21 ⟨correlations⟩
22 ⟨correlation set="atmInteraction" initiate="no"/⟩
23 ⟨correlation set="customerInteraction" initiate="yes"/⟩
24 ⟨/correlations⟩
25 ⟨scope variableAccessSerializable="no"⟩
26 ⟨variables⟩
27 ⟨variable name="loggedon" type="xsd:boolean"/⟩
28 ⟨/variables⟩ ⟨sequence⟩ ⟨assign⟩ ⟨copy⟩
29 ⟨from expression="true()"/⟩
30 ⟨to variable="loggedon"/⟩
31 ⟨/copy⟩ ⟨/assign⟩
32 ⟨while condition="bpws:getVariableData('loggedon')"⟩
33 ⟨pick createInstance="no"⟩
34 ⟨onMessage partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="logoff" variable="logoffReq"⟩
35 ⟨correlations⟩
36 ⟨correlation set="customerInteraction" initiate="no"/⟩
37 ⟨/correlations⟩
38 ⟨assign⟩ ⟨copy⟩
39 ⟨from expression="false()"/⟩
40 ⟨to variable="loggedon"/⟩
41 ⟨/copy⟩ ⟨/assign⟩ ⟨/onMessage⟩
42 ⟨onMessage partnerLink="atmFrontEnd" portType="tns:atmServicePT" operation="deposit" variable="operationReq"⟩
43 ⟨correlations⟩
44 ⟨correlation set="customerInteraction" initiate="no"/⟩
45 ⟨/correlations⟩ ⟨sequence⟩ ⟨assign⟩ ⟨copy⟩
46 ⟨from expression="20"/⟩
47 ⟨to variable="operationRsp" part="balance"/⟩
48 ⟨/copy⟩ ⟨/assign⟩
49 ⟨reply partnerLink="atmFrontEnd" portType="tns:atmServicePT" op-era-tion="deposit" variable="operationRsp"/⟩
50 ⟨/sequence⟩ ⟨/onMessage⟩
51 ⟨onMessage partnerLink="atmFrontEnd" portType="tns:atmServicePT" op-eration="withdraw" variable="operationReq"⟩
52 ⟨correlations⟩
53 ⟨correlation set="customerInteraction" initiate="no"/⟩
54 ⟨/correlations⟩ ⟨sequence⟩ ⟨assign⟩ ⟨copy⟩
55 ⟨from expression="10"/⟩
56 ⟨to variable="operationRsp" part="balance"/⟩
57 ⟨/copy⟩ ⟨/assign⟩
58 ⟨reply partnerLink="atmFrontEnd" portType="tns:atmServicePT" op-era-tion="withdraw" variable="operationRsp"/⟩
59 …
```

Fig.4 BPEL of ATM process

Table 1 Def-Use chains for ATM transaction

Service	Variable	Test level	Def-Use chains
ATM	sessionID	Service cluster	⟨2,5⟩
		Service	⟨5,9⟩, ⟨9,22⟩
		Activity	⟨5,7⟩, ⟨9,20⟩, ⟨9,34⟩
	Connected	Service	⟨0,12⟩, ⟨12,(14,15)⟩
	Customerid	Service	⟨0,17⟩, ⟨17,23⟩
	customerName	Service	⟨0,20⟩, ⟨0,36⟩, ⟨0,44⟩, ⟨0,53⟩
		Activity *r*	⟨0,34⟩, ⟨0,42⟩, ⟨0,51⟩
	Loggedon	Service	⟨27,30⟩, ⟨30,(32,33)⟩, ⟨30,40⟩
	Amount	Activity	⟨0,42⟩, ⟨0,51⟩
	Balance	Activity	⟨0,47⟩, ⟨47,49⟩, ⟨47,56⟩, ⟨56,58⟩
ATMSupport	newSessionRsp. sessionID	Service cluster	⟨0,2⟩
		Service	⟨2,4⟩

5 Conclusion and Future Work

This paper proposes a test method for BEPL-based Web Service composition based on data flow analysis to uncover the Web Service composition defects. The test model is presented to capture the data flow test artifacts of Web Service composition from three perspectives, and three levels. This test model consists of two parts: service model and structure model. The service model crosses the provider/requestor boundaries. The three components in service model, request implementation, Web Service finder, activity, and their relationships are discussed for Web Service composition based on the characteristic of Web Service. Then the structure model is introduced based on the three classified components in service model, and three types of flow graphs are used to extract the data flow information.

Based on the method proposed in this paper, the prototype system is developed, which can efficiently generate test cases for Web Service composition testing based on BPEL. Moreover, a study involving the semantic analysis for Web Service composition testing to flexibly test the Web Service composition is being investigated.

References:

[1] Business process execution language for Web services, Version 1.1. 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/

[2] Bratt S. Web services: Challenges and solutions. 2005. http://www.w3.org/2005/Talks/0420-sb-gartnerWS/slide2-1.html

[3] He H. What is service-oriented architecture. 2003. http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[4] Bloomberg J. Web services testing: Beyond SOAP. ZapThink LLC, 2002. http://www.zapthink.com

[5] Gordon AD, Pucella R. Validating a Web service security abstraction by typing. In: Proc. of the 2002 ACM Workshop on XML Security. New York: ACM, 2002. 18−29.

[6] Sumra R, Venkatvaradan R. Web service's test harness: A functional, load, and performance testing framework for Web services. http://www.developer.com/services/article.php/2229161

[7] Yang YP, Tan QP, Xiao Y. Verifying Web services composition based on hierarchical colored Petri nets. In: Proc. of the 1st Int'l Workshop on Interoperability of Heterogeneous Information Systems. New York: ACM, 2005. 47−54.

[8] Foster H, Uchitel S, Magee J, Kramer J. Model-Based verification of Web service compositions. In: Proc. of the 18th IEEE Int'l Conf. on Automated Software Engineering. Washington: IEEE Computer Society, 2003. 152−161.

[9] Fu X, Bultan T, Su JW. Analysis of interaction Web services. In: Proc. of the 13th Int'l World Wide Web Conf. New York: ACM, 2004. 621−630.

[10] Narayanan S, Mcllraith SA. Simulation: Verification and automated composition of Web services. In: Proc. of the 13th Int'l World Wide Web Conf. New York: ACM, 2002. 77−88.

[11] Ball T, Rajamani SK. Automatically validating temporal safety properties of interfaces. In: Proc. of the 8th Int'l SPIN Workshop on Model Checking of Software. New York: Springer-Verlag, 2001. 103−122.

[12] Lomuscio1 A, Qu HY, Sergot M, Solanki M. Verifying temporal and epistemic properties of Web service compositions. Berlin: Springer-Verlag, LNCS, 2007. 456−461.

[13] Fu X, Bultan T, Su J. Model checking interactions of composite Web services. 2004. http://www.cs.ucsb.edu/~su/tmp/Map2SPIN.pdf

[14] Rapps S, Weyuker EJ. Selecting software test data using data flow information. IEEE Trans. on Software Engineering, 1985,11(4): 367−375.

[15] Hong HS, Cha SD, Lee I, Sokolsky O, Ural H. Data flow testing as model checking. In: Proc. of the Int'l Conf. on Software Engineering. 2003. 232−242.

[16] Liu CH. Data flow analysis and testing of Java server pages. In: Proc. of the 28th Annual Int'l Computer Software and Applications Conf.—Workshops and Fast Abstracts. Washington: IEEE Computer Society, 2004. 114−119.

[17] Li Y, Peng CW, Liu YJ. Research and realization of networked method for structural test. Test Technology and Testing Machine, 2007,47(1):9−13.

[18] Karam MR, Smedley TJ. A data-flow testing methodology for a dataflow based visual programming language. In: Proc. of the Human Centric Computing Languages and Environments. Washington: IEEE Computer Society, 2002. 86−88.

[19] Zhao JJ. Data-Flow-Based unit testing of aspect-oriented programs. In: Proc. of the Computer Software and Applications Conf. Washington: IEEE Computer Society, 2003. 188−197.

[20] Chen JF, Shen JY, Wang XJ, Liu Y, Wang ZH. Automatic test data generation algorithm based on data flow rules. Microelectronics & Computer, 2007,24(1):5−9.

[21] Chen WH, Lu CC. Executable test sequence for the protocol control and data flow property with overlapping. In: Proc. of the IEEE Symp. on Computers and Communications. Washington: IEEE Computer Society, 2002. 251−257.

[22] Dong WL, Meng LM. tML schema based ICS proforma and generation method. Chinese Journal of Electronics, 2005,14(4): 681−685.

[23] Qadeer S, Rajamani SK, Rehof J. Summarizing procedures in concurrent programs. ACM SIGPLAN Notices, 2004,39(1):245−255.

[24] Dong WL, Yu H, Zhang YB. Testing BPEL-based Web service composition using high-level Petri nets. In: Proc. of the 10th IEEE Int'l Enterprise Distributed Object Computing Conf. Washington: IEEE Computer Society, 2006. 441−444.

DONG Wen-Li was born in 1976. She is an assistant professor at the Institute of Software, the Chinese Academy of Sciences. Her current research areas are Web technology and software engineering.

HU Jian-Hua was born in 1966. He is an associate professor at Huaihai Institute of Technology. His researches areas are computer application and network security.