

OBJECT-PROCESS DIAGRAMS AS EXPLICIT GRAPHIC TOOL FOR WEB SERVICE COMPOSITION

Yin Liu

Tongji Center of National Engineering Research Center for High Performance Computer,
Department of Computer Science, Tongji University, Shanghai, P.R. of China

Liu Wenyin

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Hong
Kong SAR, P.R. of China

Changjun Jiang

Tongji Center of National Engineering Research Center for High Performance Computer,
Department of Computer Science, Tongji University, Shanghai, P.R. of China

Web Service composition has become increasingly important with more and more Web Services developed and deployed on the Web. It requires a formal and clear representation for analysis, design, and implementation. The Business Process Execution Language for Web Service (BPEL4WS), which is an XML-based language, provides a good basis to describe Web Service composition. However, it still lacks a formal, explicit and graphic representation for visual modeling of the composition process and result. The Object-Process Methodology (OPM) has been shown to successfully describe the structure and behavior of systems using an integrated and coherent set of Object-Process Diagrams (OPDs). It should also be suitable to describe Web Service composition. In this paper we will discuss the extensions which are necessary to OPM in order to describe Web Service composition. We propose several mapping rules between BPEL4WS and OPD that are identical to the rules between Object-Process Language (OPL) and OPD so that the OPD set can be automatically created from existing BPEL4WS documents and BPEL4WS documents can be generated from the OPD set automatically, with some manual work. With the visual and explicit representation of Web Service composition, it will make the design and implementation of the composed services easier and more comprehensive.

Keywords: *Object-Process Methodology, Web Service Composition.*

1. Introduction

Web Services are self-contained, self-describing software components which can be published, discovered and invoked across the Web. The objective of developing and using Web Services is to achieve universal interoperability between applications by using Web standards. Web Services use a platform-neutral protocol and a loosely-coupled model in order to allow flexible integration of heterogeneous systems in a variety of domains including B2C, B2B, as well as enterprise applications. The building blocks of the platform-neutral protocol are Web Service Description Language (WSDL), Universal Description, Discovery and Integration (UDDI), and the Simple Object Access Protocol (SOAP), which are all based on XML language. The loosely-coupled model, which we want to discuss in this paper, is Web Service composition.

As the capability of an individual Web Service is limited, it is necessary to create new functionalities with existing Web Services in the form of processes or flows. Web Service composition is, as explained in (Casati, *et al.*, 2000 and Piccinelli, 1999), the ability to take existing services (or building blocks) and combine them to form new services. Though WSDL has provided a model of synchronous or asynchronous interactions for Web Service composition, the real application of Web Service composition is more complex than that. In order to solve this problem, several standards have been proposed.

Web Service Flow Language (WSFL, 2001) and XLANG (XLANG, 2001) are two of the earliest languages to define the standards for Web Service composition. Both of them extend WSDL and provide their own models for Web Service composition. WSFL is proposed as an XML-based language and uses Flow Model and Global Model to describe complex service composition. XLANG extends WSDL with behavioral specifications to provide a model for orchestration of services. Later, Business Process Execution Language for Web Service (BPEL4WS, 2002) is proposed as a convergence of the ideas in XLANG and WSFL specifications. It combines the best of both WSFL (e.g., support for concurrent processes) and XLANG (e.g., structural constructs for processes) into one cohesive package that supports the implementation of any kind of business process in a very natural manner. Hence, in this paper, we focus our discussion on BPEL4WS.

The representation of a composed Web Service using BPEL4WS is a set of XML documents, which are usually very long and cannot be understood easily. After several case studies on BPEL4WS were done, we found that there is little work on the graphic representation of Web Service composition and that it is necessary to find a graphic tool to visually model the composition work in BPEL4WS. This is the motivation of our OPD-based approach. With the help of the formal and explicit representation of OPDs (which will be explained in detail later), it is easy for us to understand the underlying mechanism of the composed Web Service and to accurately implement the final system.

Object-Process Methodology (OPM) (Dori, 1995 and 2002) is a framework for system modeling and engineering, which has been shown to successfully describe the structure and behavior of systems within an integrated and coherent set of Object-Process Diagrams (OPDs) (Dori, 1995 and 2002). OPM includes a clear and concise set of symbols that form a language enabling the expression of the system's building blocks as well as their relationship to each other. In other words, it is a symbolic representation of the objects in a system and the processes they enable. Objects and processes, collectively referred to as "things", are the two types of OPM's universal building blocks. With these two basic building blocks, OPD can serve as a formal and explicit graphic tool or language (actually, a visual programming language) for modeling the concurrent system in the world of Web Service composition.

Generally speaking, our approach to Web Service composition is based on OPD and BPEL4WS. We use BPEL4WS as the language to specify composed processes and OPD as the graphic tool to visualize the implementation framework of the final system. The major work in the paper is to define several OPD templates for BPEL4WS elements, which are XML tags with specific semantics. An OPD template is a pre-defined sub-OPD which is used to describe a specific control flow. Some parts of an OPD template are defined as variables and require modification when it is used in practice. With these templates it is possible for us to automatically create OPDs from existing BPEL4WS documents for better understanding, or to design a composed service in OPDs and then generate BPEL4WS documents from them.

Specifically, an equivalent link, which is a new extension to OPD, is introduced in this paper. It expresses an equivalence relationship between two things (objects or processes). Its semantics will be explained in the overview of OPD and detailed usages will be presented when we introduce OPD templates for BPEL4WS elements.

In the following sections, we will first give an overview of BPEL4WS and OPD as background material. Then, several mapping rules between OPD and BPEL4WS will be presented, in which a set

of OPD templates that represent elements in BPEL4WS is defined. Based on these templates, an example in the real world will be presented to illustrate the application of our OPD-based approach to Web Service composition. Finally, we will present a concluding remark to summarize our work.

2. Background

In this section, we will give an overview of BPEL4WS and OPD. We will first introduce BPEL4WS briefly and divide its elements into 5 categories, which will be analyzed one by one in the next section. Then the concepts and symbols of OPD will be introduced and a table of graphic symbols will be presented that we will use for Web Service composition.

2.1. Overview of BPEL4WS

Business Process Execution Language for Web Service (BPEL4WS, 2002) is first proposed by BEA, IBM, and Microsoft. It consists of a set of XML tags, which define the notation for specifying process behaviors based on Web Services. BPEL4WS provides an XML-based language for the formal specification of business processes and business interaction protocols. In this section, all BPEL4WS elements are classified into 5 categories, as shown in Table 1, for better understanding and analysis.

Table 1 Categories of elements in BPEL4WS

Category	Elements	Category	Elements
Structural Control Elements	<process>	Data Structure Elements	<property>
	<scope>		<propertyAlias>
	<flow>		<correlationSets>
	<sequence>		<correlations>
	<receive>		<containers>
	<pick>	Service Related Elements	<serviceLinkType>
	<reply>		<serviceReference>
	<switch>		<partners>
	<while>	Exception Handling Elements	<faultHandlers>
<links>	<compensateHandler>		
Functional Elements	<invoke>		<compensate>
	<assign>		<throw>
	<wait>		<catch>
	<empty>	<catchAll>	
	<terminate>		

- 1) Structural control elements are the building blocks of control flow of the composed service. <process> is the top-level element which represents a composed service. <scope> is used to define a domain with its own exception and compensation handler. <flow> and <sequence> are activity containers, in which activities can be executed concurrently or in sequence of definition. <receive> and <pick> can wait for one or several messages on operations of specific ports. When <receive> waits for a request-response operation, the corresponding response is returned by <reply>. This pattern indicates a synchronous interaction between the partners and the composed

service. The <switch> element provides a mechanism of conditional branching. The <while> element presents a while-do loop for iteration. Because the activities in the control flow can be executed concurrently, a synchronization mechanism is required and supported by the <links> element.

- 2) Data structure elements allow us to preserve the states of current interaction and define the input and output data structures for each operation. <property> defines a global unique name of a token for correlation of service instance with messages. Through the <propertyAlias> element, a <property> is mapped to a part of a specific message. When a business process is launched, each Web Service involved should create an instance for itself. Because there will be several instances on the same Web Service, the messages sent to it should be distinguished and delivered to corresponding instances. <correlationSets> and <correlations> provide a common mechanism for it. <correlationSets> defines a set of properties whose values identify an instance on a Web Service. <correlations> is used to pass this identification in the interaction with external Web Services. The <container> element can be regarded as a global value of a specific message type, which will be used as the input and output when invoking an operation of an external Web Service.
- 3) Functional elements provide some useful functions for Web Service composition. With the <invoke> element, an operation of an external Web Service can be invoked. The <assign> element is used to do some simple calculations and set the values of the containers. The <wait> element provides a mechanism for event and timeout. <empty> is an activity which does nothing. The <terminate> element will stop the execution of a current instance.
- 4) Service-related elements describe the external requirements of the composed Web Service, which is important for deployment. <serviceLinkType> consists of one or two port types of related Web Services. It is used to define a <partner> in the composed service. The <partner> element also defines the role of the composed service. The <serviceReference> element is used to describe an external Web Service. Since these notions are just preliminary and a standard for them does not exist, they will not be further elaborated on later in detail.
- 5) Exception handling elements describe the exception and compensation mechanism in BPEL4WS. Exception is introduced into BPEL4WS as a formal method to handle fault messages returned from an operation. <faultHandlers> can be installed into several elements to handle the exceptions which occur inside it. <catch> and <catchAll> elements are enclosed in <faultHandler> to catch specific exceptions. <throw> is used to throw an exception in certain situations. When an exception occurs in an activity, the job, which has been finished successfully in the activities that are enclosed in the current activity, should be rolled back and some compensation operations, which are defined in the business logic, should be taken for these activities. These operations are defined by <compensateHandler>. The <compensate> element is used to invoke compensation operation of an activity that has already completed its execution normally.

2.2. Overview of OPD

The Object-Process Methodology (OPM) (Dori, 1995 and 2002) is a system analysis and design approach that combines ideas from OOA and DFD within a single modeling framework to represent both the static/structural and dynamic/ procedural aspects of a system in one coherent frame of references. The use of a single model eliminates the integration problem and provides clear understanding of the system being modeled. The object-process diagram (OPD) (Dori, 1995 and 2002, Liu and Dori, 1999) is OPM's graphic representation of objects and processes in the universe of interest along with the structural and procedural relationships that exist among them. Due to synergy, both the information content and expressive power of OPDs are greater than those of Data Flow Diagram (DFD) and Object-Oriented Analysis (OOA) diagrams combined.

In OPM, both objects and processes are treated analogously as two complementary classes of things, elementary units that make up the universe. The relationships among objects are described using structural links, such as aggregation-participation and generalization-specialization. The relationships between objects and processes are described by procedural links, which are classified into effect, agent, and instrument links. Symbols of them are illustrated in Fig. 1.

A very important feature of things (objects and processes) in OPDs is their recursive and selective scaling ability, which provides complexity management through control of the visibility and resolution of the things in the system. In general, things are scaled up (zoomed in) as we proceed from analysis to design, and then to implementation. The scaling capability provides for function definitions and calls. Specifying generalization-specification among processes enables the establishment of inheritance relations among processes in a manner similar to inheritance among objects.

A thing is the elementary unit that makes up the universe. An object is a persistent, unconditional thing. A process is a transient thing, whose existence depends on the existence of at least one object. These terms are originally proposed for system analysis in OPM (Dori, 1995 and 2002). From the design and implementation viewpoint, an object can be regarded as a variable with a specified data type, while a process is a function or a procedure operating on the variables, which are objects. A state of a thing at a given point in time is the set (or vector) of attribute values the thing may have at that point in time. Object, process and status together are referred to as entities in OPM.

Certain structural relations between two objects, namely Aggregation-Participation, Featuring-Characterization, and Generalization-Specialization, collectively referred to as the *fundamental relations*, are represented by a triangular symbol along the link that connects them. Aggregation-Participation describes the relationship of composition between two objects. The meaning of Featuring-Characterization's follows its name: It is the relation between a feature - an attribute or an operation ("method", "service") and the thing that the feature characterizes. The Generalization-Specialization link between two objects induces an inheritance relationship between two object classes. The Instantiation link indicates an object is an instance of a class.












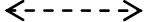
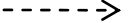
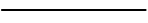
Things		Structural Relations		Procedural Links	
Object		Aggregation-Participation		Agent link	
Process		Featuring-Characterization		Instrument link	
State/Value		Generalization-Specialization		Effect link	
		Instantiation		Consumption/ result link	
		Equivalent link*		Control link	
		Data link			

Fig. 1 OPD symbol set for Web Service composition

In this paper, a new link is proposed, referred to as equivalent link, as an extension to OPD. An equivalent link describes two kinds of equivalence: the equivalence between processes and the equivalence between objects. The equivalence between processes can be regarded as a proxy-implementation relationship between them, which means that one process is the implementation of the function and the other is just a proxy of it. The equivalence between objects means one object is just an alias of the other, and their value should be deemed unique.

Agents and instruments are enablers of processes. They exist before the process execution and their state (set of attribute values) is not altered by the process execution. An effect link connects an affected object to the affecting process. An affectee is an object whose state is altered by the process. A consumed object is an object that is consumed (and destroyed) by the process and no longer exists after the process execution. A resulting object is a new object constructed as a result of the process execution. The consumption link is graphically represented by a one-way arrow, directed from the consumed object to the consuming process. The result link is also represented by a one-way arrow, but the arrow in this case is directed from the process to the resulting object. The effect link is represented by a two way (bi-directional) arrow between the affected object and the process.

3. Mapping Rules between OPD and BPEL4WS

In this section, we will present several mapping rules between OPD and BPEL4WS, which focus on how to represent elements in BPEL4WS using the OPD approach. Several specific OPD templates, which are pre-defined sub-OPDs used to describe specific control flows, will be introduced in order to describe the structural control elements in BPEL4WS. Some parts of an OPD template are defined as variables and require modification when it is used in practice. We will also propose an equivalent link as an extension for OPD, which is necessary for representing certain elements of BPEL4WS by using OPD. With these mapping rules, an OPD set can be easily created from existing BPEL4WS documents.

3.1. Structural Control Elements

The <process> and <scope> elements are used to define a domain which has its own fault and compensation handler. <process> is the largest domain in the composed service and will be regarded as the top-level OPD. Each time when a <scope> element is required, a new process needs to be inserted, representing this scope, into the current OPD and a new OPD for it needs to be created. We call this process SCOPE template.

Both <flow> and <sequence> elements are activity containers which hold a set of activities and specify the invoking sequence of these activities. The activities, which are enclosed in a <flow> element, can be executed concurrently, whereas the activities, which are enclosed in a <sequence> element, can only be executed as defined by the sequence. FLOW and SEQUENCE templates of OPDs are used to represent them, respectively. In a FLOW template all processes are laid on the same level, but in a SEQUENCE template, the control link (Liu and Dori, 1999) should be introduced to present sequential relationship between processes. Fig. 2 illustrates these two templates.

<receive>, <pick> and <reply> are the elements which provide the message-exchanging methods of a composed service. The <receive> element represents a process which waits for a specific message. The <pick> element represents a process which waits for a set of messages and executes a specific activity for each message. Once a message is received in <pick>, all other waiting operations are canceled. The <reply> element determines output of the composed service. The RECEIVE, PICK, and REPLY templates are used for each of them. Fig. 3 illustrates these three templates. In Fig. 3 OP represents an operation of an external Web Service, and P1, P2 ... Pn represent sub-processes in the PICK template. These abbreviations will also be used in the following figures. In order to simplify the representation, in Fig. 3 the correlation issue is not introduced, which is important and related to these elements and will be further detailed in the discussion of the <invoke> element.

Here we introduce the first usage of an equivalent link as a way to invoke an operation of a port for an external Web Service. The graphic representation of an equivalent link is a dashed line with arrows on both ends. The semantics of the equivalent link when it connects two processes is that one process is the implementation and the other is the proxy. In the world of Web Services, it means that the operations of these two processes are “dual,” as explained in (WSFL, 2001), that one is solicit-response

operation and the other is a request-response operation, or one is a notification operation and the other is a one-way operation.

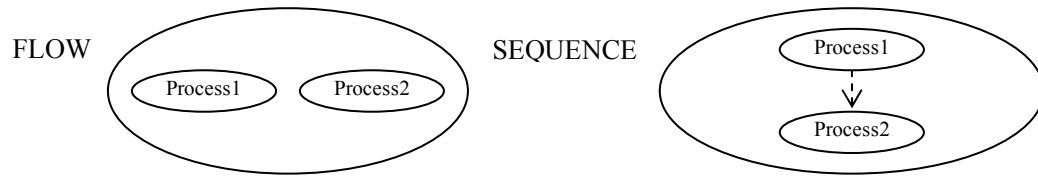


Fig. 2 FLOW and SEQUENCE templates

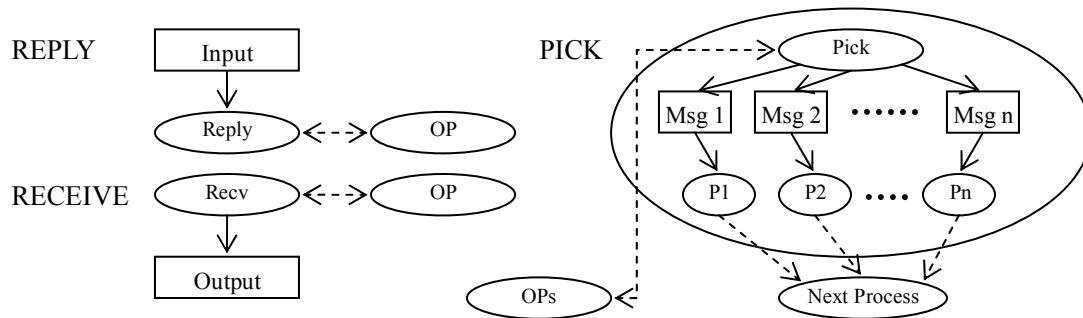


Fig. 3 REPLY, RECEIVE, and PICK templates

Both <switch> and <while> elements are used for flow control in BPEL4WS. The <switch> element provides a mechanism of conditional branching, whereas the <while> element presents a while-do loop for iteration. We use SWITCH and WHILE templates in Fig. 4 to represent them, respectively. C1, C2 ... Cn are conditions for all cases involved in the <switch> element.

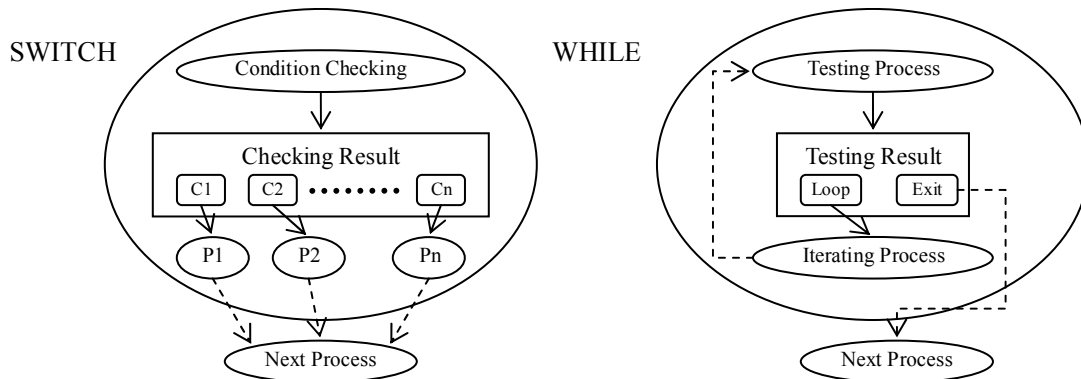


Fig. 4 SWITCH and WHILE templates

The last but most important element in flow control is the <links> element. Recalling that the activities in the <flow> element can be executed simultaneously, it is natural that a synchronization

mechanism is required. The <links> element works with the standard elements of an activity to achieve this objective. If an activity is the source of a link, it probably has a transition-condition which indicates whether the link is enabled or disabled. Each activity will have a join-condition which determines whether this activity should be executed or skipped. We use an ACTIVITY template in Fig. 5 to wrap any activity which is the source or target of a link in the composed service.

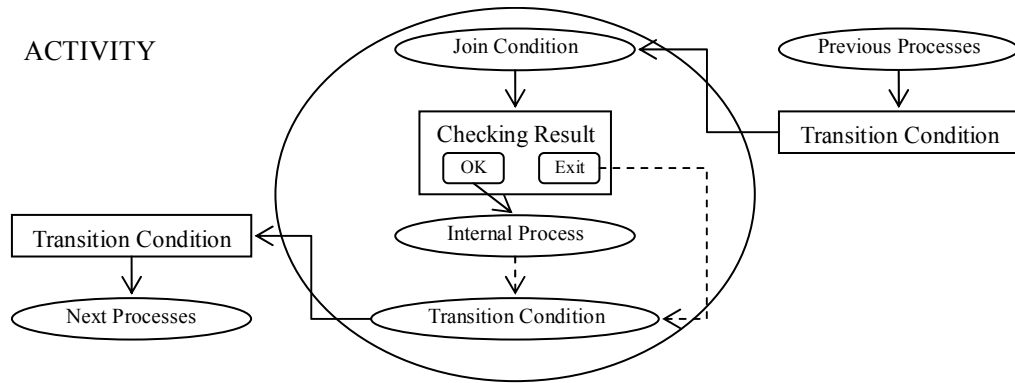


Fig. 5 ACTIVITY template

3.2. Data Structure Elements

All data structures in BPEL4WS, including message, property, container, and correlation set, can be represented as objects and structural links of OPD. A message defined in WSDL with a group of data members is represented with an aggregation-participation structural link in OPD; and a container defined in BPEL4WS, which is a global variable of specific message type defined in WSDL, can be regarded as an instance of a message type, which is represented with an instantiation structural link in OPD. See the following example in Fig. 6.

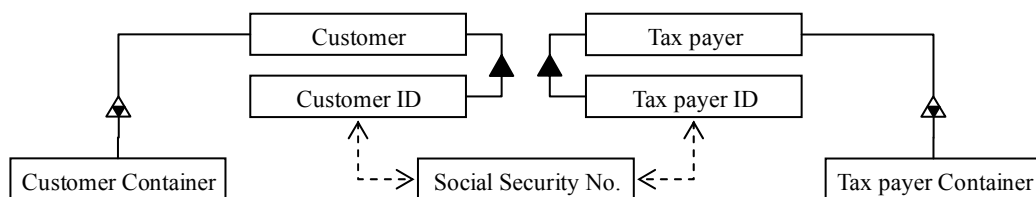


Fig. 6 Example for message, container, and property alias

<property> and <correlationSets> are often used together to represent instance related data. For example, a social security number, which identifies an individual tax payer, is an instance-related data in a long-running multiparty business process regarding a tax matter. It may appear in various parts of different messages, but in a specific instance, the value is unique. <property> defines a global unique name of a token for correlation of service instance with messages. Through the <propertyAlias> element, a <property> is mapped to a part of a specific message.

Here the second usage of the equivalent link is introduced as a way to present equivalence between parts of different messages, which means that the related parts in different messages should be

identical. The example in Fig. 6 demonstrates such a scenario. Social security number, which is unique and used to distinguish messages between instances, is mapped to two parts of different messages.

Each correlation set in BPEL4WS is a named group of properties that serve to define a way of identifying an application-level conversation within a business protocol instance. A given message can carry multiple correlation sets. A correlation set can be used in the <correlations> element which appears in the <invoke>, <receive>, <reply> and <pick> elements. According to the pattern and initialization flag of the <correlations> element, the properties in a correlation set are copied to or from the output or input messages. A detailed example of correlation will be explained in the discussion of the <invoke> element in the next section.

3.3. Functional Elements

The <invoke> element is used to interact with external Web Services. An INVOKE template is used to represent it. An equivalent link is used here to represent the “dual” relationship between the process and the external Web Service. The INVOKE template can be regarded as a sub-process of RECEIVE, REPLY, and PICK templates because it introduces the correlation issue which is not discussed in the above sections. We should use an INVOKE template in Fig. 7 to replace the Receive, Reply, and Pick processes in the above templates in practice.

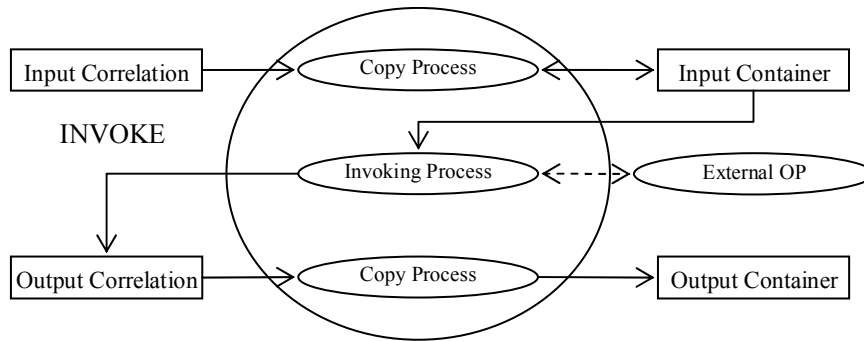


Fig. 7 INVOKE template

The <assign>, <wait>, <empty> and <terminate> elements are used to provide some specific functions; <assign> provides the functions to set the value of a part of specific container. <wait> is a process which will wait for a given time-period or until a certain time has passed. <empty> just represents a null process and the execution of the process of <terminate> will terminate execution of the entire instance. We just provide a simple template for each one in Fig. 8.

3.4. Service Related Elements

The port and operation definitions in WSDL are already sufficient to generate OPDs. Each port type defined in WSDL will be regarded as an object which exhibits several operations. The exhibition-characterization link is used to represent it. Fig. 9 illustrates an example.

As stated in (BPEL4WS, 2002), the notions of service link and service reference are preliminary and there are no standards for them. So we do not discuss them in detail in our approach. In order to convert BPEL4WS documents to OPD, we just regard a partner and its related service link as an object which is composed of one or two port objects.

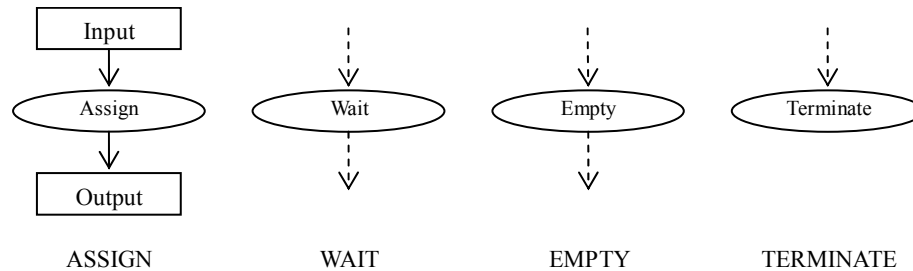


Fig. 8 ASSIGN, WAIT, EMPTY, and TERMINATE templates

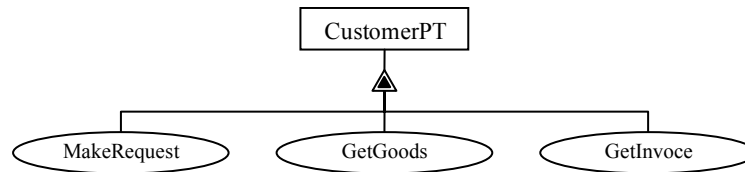


Fig. 9 Example for port definition in WSDL

3.5. Exception Handling Elements

Exception handling elements are used to handle errors in the composed service. The semantics of these elements is similar to structured exception handling statements in C++, except for <compensationHandler> and <compensation> elements, which are used to support user-defined transaction mechanisms. Frankly, there is no appropriate and formal way in OPD to describe these elements. We will cover this issue in our further work.

4. The OPD-based Approach to Web Service Composition

In the above section, a set of mapping rules between OPD and BPEL4WS are introduced. With these mapping rules and corresponding OPD templates, we propose our OPD-based approach to Web Service composition. The approach could be divided into three steps: First, OPD is used to describe the global flow of the composed service exactly like in the ordinary OPM analysis and design process. Second, when we delve deeper and design the detail of the process, the OPD templates introduced in the above section can be used to build the control flow. Finally, when all OPDs are created, corresponding BPEL4WS documents can be generated from them.

In the rest of this section, we will present an example to illustrate how an OPD-based approach to Web Service composition is used in practice. The example is taken from (BPEL4WS, 2002) and a few modifications are made to it. Though the example is rather simple, it is sufficient to illustrate the expressive power of our OPD-based approach.

4.1. Case Description

The case is about a shipping service which handles shipment orders. From the view of the shipping service, an order is composed of the name of item, the destination, and the number of items. The shipping service offers two kinds of shipments: (1) shipments in which the items are held and shipped together and (2) shipments in which the items are shipped piecewise until all of the items are processed.

Four roles are involved in the process, the customer, the shipping service, the checking service and the shipment handler. The customer should place a shipment order and send it to the shipping service, then wait for shipment notice. The interactions between customer and shipping service are asynchronous. After the order is received, the credit of the customer and the possibility to handle this order should be checked by the checking service. The interactions between shipping service and checking service are synchronous. Before the shipping service sends a shipment notice back to the customer, it should ask the shipment handler to ship the items included in the notice. The interactions between shipping service and shipment handler are synchronous.

Assuming that Web Services for customer, checking service, and shipment handler are already implemented, our job is to compose the shipping service.

4.2. Global Flow

According to the above description, the global flow is illustrated in Fig. 10. The global flow of the shipping service is composed of three processes. The first process is waiting for the shipment order from the customer. The second process is checking the credit of the customer and the possibility to handle this order. The third process is handling the shipment order. Shipment order and checking result are messages exchanged among them. SEQUENCE and FLOW templates are used here to indicate the execution sequence.

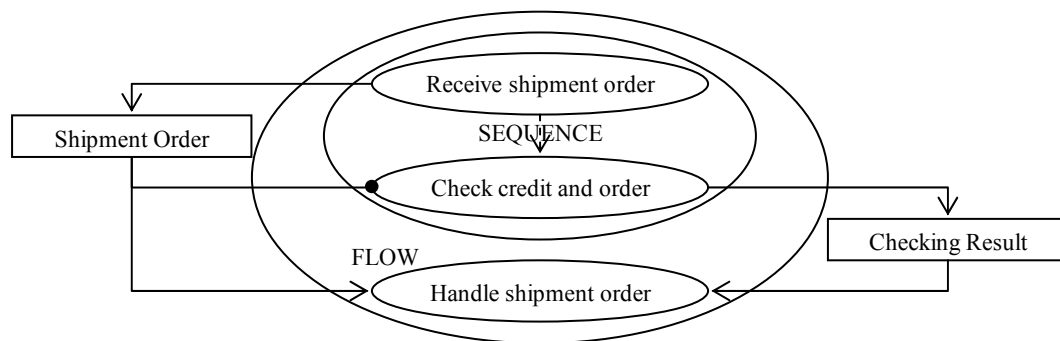


Fig. 10 Global flow of shipping service

4.3. Detailed Design

In the detailed design, we first present the OPDs for definition of ports and messages. In Fig. 11, there are three kinds of messages existing in the system. ShipNoticMsg and ShipOrderMsg are messages which are exchanged among all roles involved. CheckingResultMsg is returned by the checking service. CreditCheckResult, ShipNotice, ShipOrder, and PossibilityCheckResult are instances of these messages.

The operations of each port and corresponding input/output messages of each operation are illustrated in Fig. 12. With the definitions in Fig. 11 and Fig. 12, the first and second processes are extended in Fig. 10. The RECEIVE template is used for the first process, while the FLOW template is used for the second. The extended OPD is illustrated in Fig. 13.

Now the last but most complex process is left. This process can proceed only if all checking results are true. We use an ACTIVITY template in Fig. 14 to perform this check and the checking results are regarded as the input of the join condition. In the process, a SWITCH template is used in case that the customer requires two kinds of services, in which the items are shipped together or shipped in pieces.

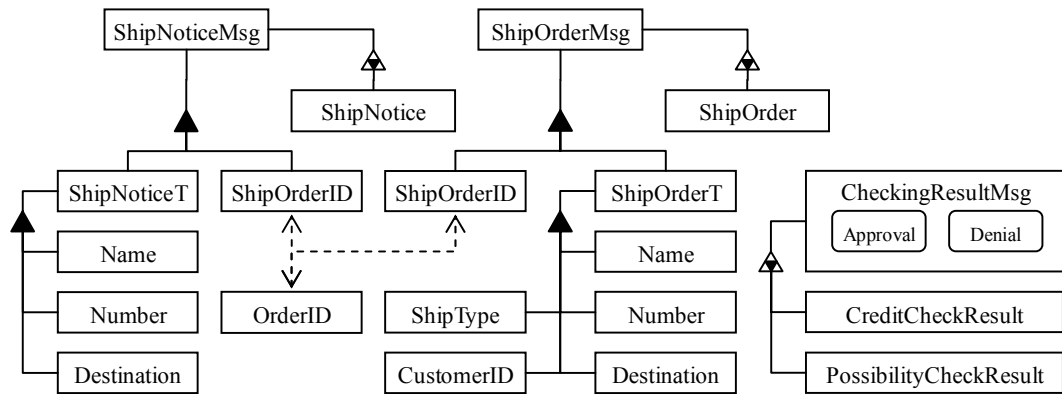


Fig. 11 Definition of messages

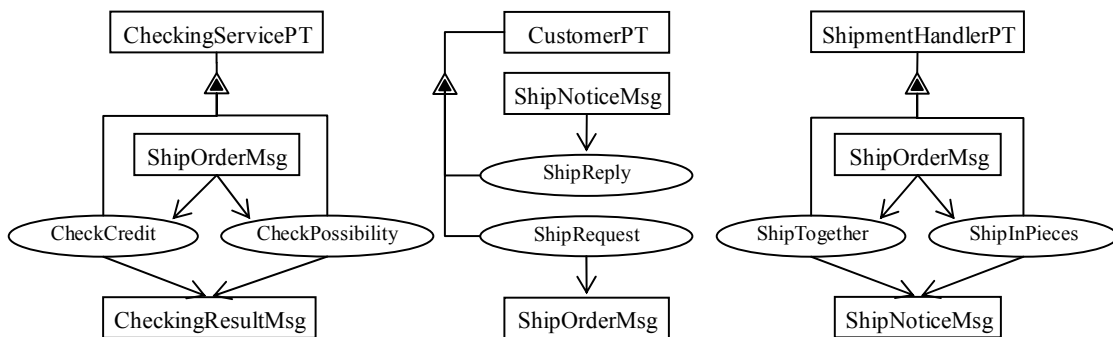


Fig. 12 Definition of ports

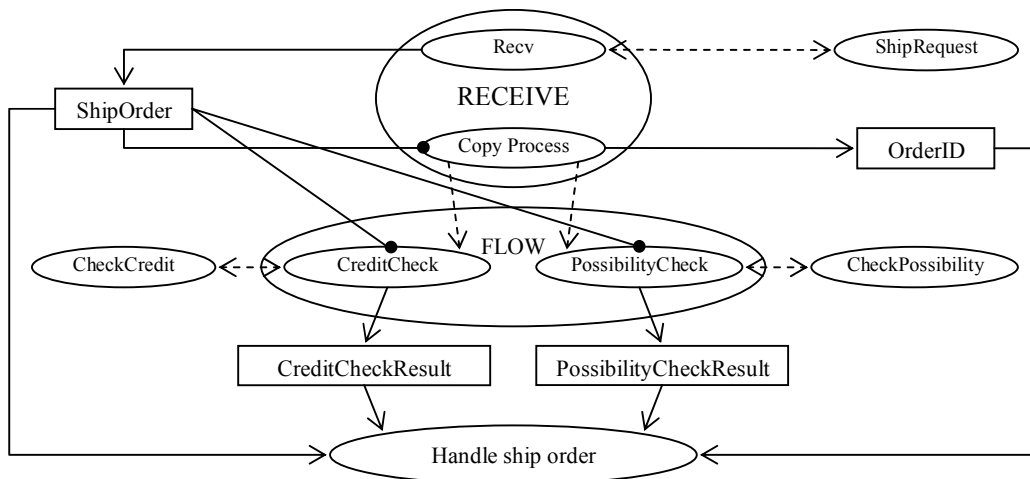


Fig. 13 “Receive shipment order” and “Check credit and order” processes are expanded

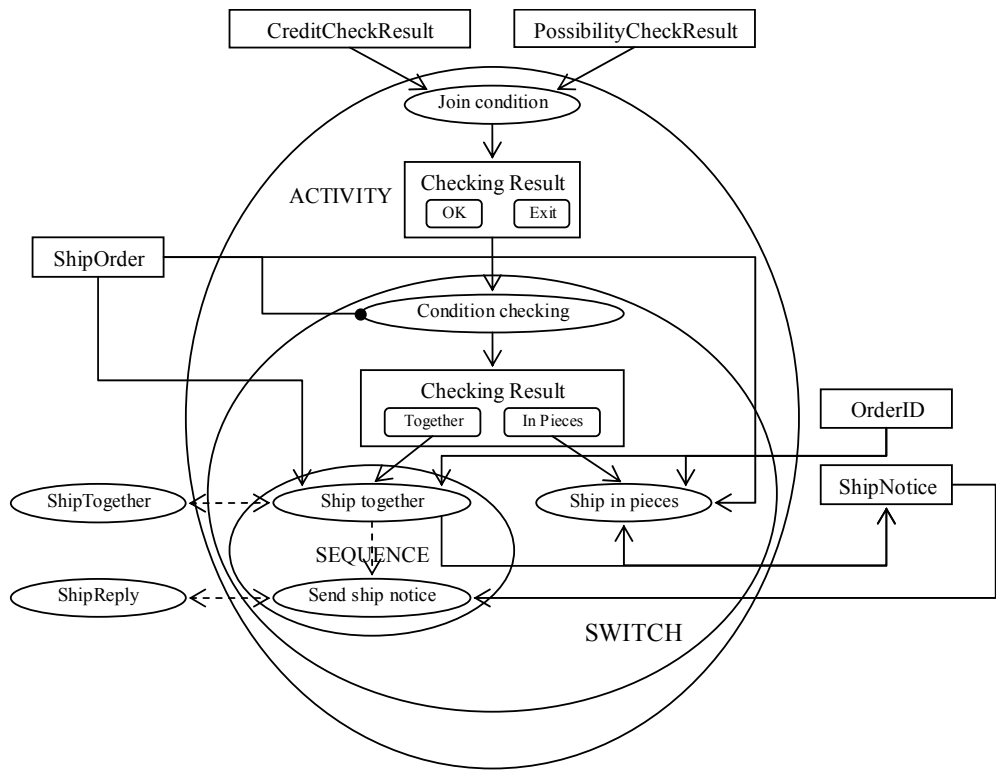


Fig. 14 “Handle shipment order” process is expanded

For the order which can be shipped in pieces, a WHILE template should be imported to fulfill the requirement. The sub-process cannot complete until all items in the order are shipped and for each piece a shipment notice is sent to the customer. Fig. 15 illustrates this sub-process.

4.4. BPEL4WS Documents

Since all OPDs are created, BPEL4WS documents according to the templates and the mapping rules are able to be generated. Frankly, some manual work is required to complete the details of the documents, but major part of documents can be generated automatically from the existing OPDs.

First of all, a WSDL definition for this shipping service should be generated, in which messages, ports, properties, property aliases, and service links are defined. The target of the WSDL definition should be specified manually. Message definitions can be generated according to the aggregation-participation structural links in Fig. 11, but the types of the objects should be defined manually. Port definitions can be generated according to Fig. 12, and the input/output of each operation can also be determined according to Fig. 12. Property and property alias definitions can also be generated according to the equivalent links in Fig. 11. Finally, service link definitions, which have only one role for each, are generated for each port. The detailed document is available at <http://www.cs.cityu.edu.hk/~liuwy/WS/A1.xml>.

Then the BPEL4WS documents for this shipping service should be generated next. A <process> element is first added, in which target and reference should be defined manually. For each port in WSDL definition, a <partner> is generated with the corresponding service link. Containers are generated according to the instantiation links in Fig. 11. Correction sets are generated according to the equivalent links in Fig. 11. Then definitions of control flows can be generated according to the

templates used in OPDs in Fig. 13, Fig. 14 and Fig. 15. Notice that in the OPDs, transition conditions, join conditions and case conditions are not defined. It is necessary to add them into the documents manually. It is also identical to the assign process, whose detail is not specified in OPDs. The detailed document is available at <http://www.cs.cityu.edu.hk/~liuwy/WS/A2.xml>.

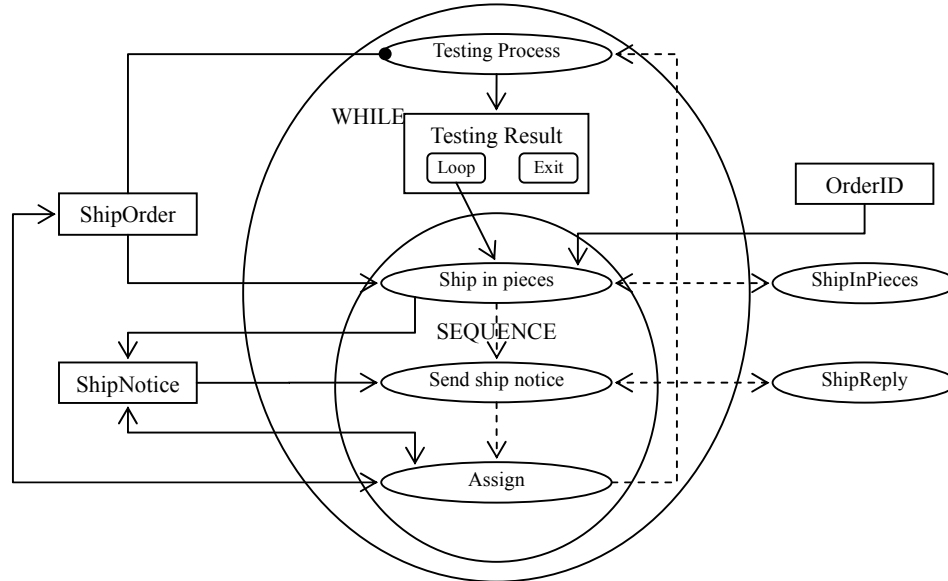


Fig. 15 “Ship in pieces” process is expanded

5. Concluding Remark

In this paper, an OPD-based approach was proposed for Web Service composition, in which several mapping rules between OPD and BPEL4WS were introduced. We extend OPD with the equivalent link and propose a set of OPD templates to represent Web Service composition originally described using BPEL4WS. The advantages of this approach include visual representation (OPDs) of the Web Service composition process, which makes it not only easier for us to understand existing BPEL4WS documents but also more convenient to design and revise Web Service composition and automatically generate BPEL4WS documents (major parts) from the visual models.

6. Acknowledgement

The work described in this paper was supported by a DAG grant from City University of Hong Kong (Project No. 7100292).

7. References

- BPEL4WS, 2002, “Business Process Execution Language for Web Service, Version 1.0”, <http://www.ibm.com/developerworks/library/ws-bpel/>.
- Casati, F., et al., 2000, “Adaptive and Dynamic Service Composition in eFlow,” Proceedings of CaiSE 2000, Stockholm, Sweden, pp. 13-31.
- Dori D., 1995, “Object-Process Analysis: Maintaining the Balance between System Structure and Behavior,” Journal of Logic and Computation, Vol. 5, pp. 227-249.
- Dori D., 2002, “Object-Process Methodology – A Holistic Systems Development Paradigm,” Springer.

Liu W. and Dori D., 1999, "Object-Process Diagrams as an Explicit Algorithm-Specification Tool," *Journal of Object-Oriented Programming*, Vol. 12, No. 2, pp. 52-59.

Piccinelli G., 1999, "Service Provision and Composition in Virtual Business Communities," Technical Report HPL-1999-84, Hewlett-Packard.

WSFL, 2001, "Web Services Flow Language (WSFL 1.0)," <http://www-3.ibm.com/software/solutions/webser vices/pdf/WSFL.pdf>.

XLANG, 2001, "Web Service for Business Process Design," http://www.gotdotnet.com/team/xml_wsspecs/xl ang-c/default.htm.