WSCE: A Flexible Web Service Composition Environment

Xiulan YU, Long ZHANG, Ying LI, Ying CHEN
{yuxl, longzh, lying, yingch}@cn.ibm.com
IBM China Research Laboratory

Abstract

In this paper, we propose the concepts of virtual partner and inspector into the web services composition. Virtual partner, as an IT level concept, is a web service (pseudo web service) using the same interface with the actual partner but different binding message. A virtual partner can be invoked directly by a business process described by BPEL, so that the BPEL programmer can test both application's functionality and non functionality performance early in the development cycle to avoid any problems in the final runtime, or test the selection of their partners in business level design. The IT virtual partners provide developers with a range of the techniques which let them explore every aspect of their program. Inspector is proposed when using the third-party process engine. An inspector itself is also a web service. The programmer can register any required output information in it. The IT virtual partner and the inspector concepts have been integrated in our WSCE, a flexible Web Services Composition Environment for a business process. WSCE is a prototype of autonomic modeling and simulation environment. With the help of a third-party BPEL engine, it provides programmer with concepts and tools to facilitate business process programming.

KeyWords: Virtual Partner, Inspector, Web service Composition, BPEL

1. Introduction

With the increasingly competitive situation in the service market, many enterprises are rushing to employ web services in to their business, such as on-line travel reservations, procurement, customer relationship management (CRM), billing, accounting, and supply chain. Currently, web services use the technologies of Universal Description, Discovery, and Integration (UDDI), Web Service Description Language (WSDL), and Simple Object Access Protocol (SOAP), and efficiently and effectively achieve sharing services on the Web. However, business integration requires more complex functionality than current web services provide. The functionality includes transactions, composition, negotiations, management, and security, et al. Web services composition is among the most attractive functions for business integration which can perform a specific task, that is, the existing services are able to cooperate although the cooperation was not designed in advance. Services composition could be static (service components interact with each other in a pre-negotiated manner) or dynamic (they discover each other and negotiate on the fly). This is potential to reduce development time and cost for new services.

Among web services composition languages, the Business Process Execution Language for Web Services (BPEL4WS) [1], the merging of IBM's Web Services Flow Language (WSFL) and Microsoft's XLANG, has the most potential to become the de facto standard. This language is based on SOAP+WSDL+UDDI basic stack. Users can compose their web services with BPEL language and run them with a BPEL engine. However, a composite Web service is in fact a system that consists of several conceptually autonomous but cooperating units. It is difficult to specify how this system would behave and ensure that it behaves as required by the specification. Usually we need two kinds of insurances: (1) the functionality character, given the input and initial state, the system may produce anticipant outputs; (2) the non functionality characters, such as quality of service (QoS), which has became increasing more important when web processes model complex and mission critical application [2]. The current solution to this problem is using model tools to model the process and QoS respectively and the results generated by the model tools are used to design the IT process and select the partners. However, this is just business level design. When an IT programmer writes the business process with BPEL, due to the complexity of the process in some applications, the composed web services may contain errors. If they are deployed to the operating system, it may lead to angry customers and loss of revenues. On the other hand, the QoS requirements modeled by modeling tools need to be verified by IT operations, even the partners in business process can be directly selected by the IT process. It is thus important to analyze business process and partner web services before they are put into operation. The goal is to provide mechanisms to support correct Web service composition.

One possible solution to this problem is to test the composed web service in true business. For example, when the developer has finished BPEL document and composite service WSDL, he can invoke partner's web services to test his service function and QoS. However, it is impossible in most business cases, partner maybe also provide service for others, he can not provide testing environment for developers; on the other side, partner may be charge service invocation, and who will pay the testing invocation is a real problem.

Another possible solution is to setup simulation environment by the testing person to help verify the web services composition. While, in this solution, testing person has to waste a lot of time to set up the simulation environment, write test program manually and deploy the service. How to make the testing automatic, in other words, the process developer just care about the process modeling and other problems will be solved by the environment, is the main problem.

In this paper, we propose the concept of virtual partner and inspector into the web services composition. In fact, the object in ER model, which can simulate the partner in a business process, is a virtual partner. But this virtual partner is in business level. Here we proposed the virtual partner in IT level: the virtual partner is a web service (pseudo web service) using the same interface with the actual partner but different binding message. The virtual partner can be invoked directly by the business process described by BPEL, so that the BPEL programmer can test both their application's functionality and non functionality performance early in the development cycle to avoid any problems in the final runtime, and also, the BPEL programmer and business man can test the selection of their partners in business level design. The IT virtual partners provide developers with a range of techniques to let them explore aspects of their program. The inspector is proposed when using the third-party process engine. The Inspector itself is also a web service; the programmer can put any output information required in it.

The IT virtual partner and inspector web service concepts have been integrated in our WSCE, a Flexible Web Services Composition Environment for a Business Process. WSCE is a prototype of autonomic modeling and simulation environment, proposed in IBM China Research Laboratory. The WSCE provides the programmer concepts and tools to facilitate business process programming with the help of a third-party BPEL engine. A developer can use this environment to model, debug and simulate the BPEL process with great flexibility.

The rest of this paper is organized as follows. Section 2 describes a motivating example. The concept of virtual partner is proposed in section 3. The inspector, as well as architecture and all components of our environment are presented in section 4 in more detail with the help of an example; Section 5 gives the experiment results to explore the feasibility of inspector web service; related work is derived in section 6, and section 7 concludes the paper and presents possible future work.

2. Motivating Example

In the paper, we use a business process integration consulting service as our motivating example. In a specific business process integration consulting service, consultant would use BPEL to integrate a business to business (B2B) service. To do this, several atomic web services provided by their partners will be composed together and be invoked according to the process defined by the consultant. One of the partners is a bank. Its every transaction will involve cash flow and it will charge every transaction. After the consultant designs the process from the viewpoint of the business strategy, it will be IT developer's responsibility to implement the process with BPEL. But he can not put his BPEL program into operation directly because: (1) his document may contain errors which will possibly cause fatal error for the service providers, so he needs to debug it; (2) the process involve cash flow and the bank doesn't know whether the transactions the designer issued is used as tests or not, so the bank will charge every transaction.

The straightforward idea is when the developer writes the BPEL program, he'd better set up a simulation environment, namely to set up a network environment, and deploys a pseudo web service for each partner, and test his BPEL program. If he wants to test different situations, he has to change the pseudo web service logic, compile and redeploy it again and again. Especially if he needs to guarantee the process QoS to customer, he has to use the model tool to evaluate whether the partner satisfies the process requirement. Usually, this is static QoS guarantee. If one partner should be selected from several partners who provide same services to meet process QoS, he must rewrite the services and redeploy them. All the operations will involve several interfaces for the BPEL developers; he has to be very familiar with all the software, which costs him a lot of time. So the developer is required to master higher skill and need much time to deploy the service, which increases the business investment. Besides, the BPEL is still a developing language for business process. When the new version of the language is provided, the engine of BPEL has to be updated simultaneity.

Therefore it is urgent for BPEL development environment to provide developers with an automation and virtualization development and simulation environment. With IT virtual partner, the BPEL programmer can focus on the process coding. When he needs to test the business process, the development environment can generate the virtual partner he needs. What he needs to do is just to input his QoS parameters. Then it will be much easier for IT engineer to obtain business goal.

3. Virtual Partner and Inspector

One unique feature of the environment is that the concept of virtual partner is introduced for the first time. The virtual partner in IT level is a web service (pseudo web service) using the same interface with the actual partner but different binding message. The virtual partner can be invoked directly by the business process described by BPEL, and the generation of virtual partners is transparent to the user. So when a developer finishes BPEL modeling, our environment can generate partner web service skeleton automatically according to partner's WSDL, and the developer can input the functionality and non functionality in the web service according to the requirement. Then the environment can deploy the service to local host container automatically. The web service is called pseudo web service and the partner is called virtual partner. Virtual partner and pseudo web services provide developer a functionality and non functionality simulation running and testing environment.

Another unique feature of the environment is the concept of inspector. Since the BPEL is still under development, our environment tries to integrate the third-party BPEL engine, which maybe provide no API for us to test the draft BPEL process. In order to make the simulation environment more reality, we proposed inspector concept to provide user inspecting interface. The inspectors are uniformly designed as web services, similar to the partner's service. The inspectors are generated and deployed to the local host container atomically in our environment. After plugging it in BPEL engine, all the components form a virtual running environment for BPEL, so that the developer can test and simulate his business process to find the possible problems without relying on the actual partners' web service.

Although this virtual environment is some like a simulation environment, it is obviously different from the current simulation environment based on discrete event simulation of ER (Entity-Relationship) model: it provides tools for developers to test their application's performance early in the development cycle so that you can avoid any problems in the final runtime. These profiling tools provide developers with a range of techniques to let them explore aspects of their program.

4. The Flexible Web Service Composition Environment

In our WSCE environment, the concepts of virtual partner and inspector have been implemented. The programmer can program BPEL process with drag and drop tool, debug and simulate the process using a third party BPEL engine in WSCE.

4.1 Architecture of the Environment

Fig.1 shows the architecture of the whole environment. The whole environment includes BPEL studio and servlet container. BPEL studio is used for both design and testing time and servlet container which contains a BPEL engine is used for runtime. BPEL studio includes drag-drop modeling tool, virtual partner generator, inspector generator, simulated BPEL generator, and result analyzer. Drag-drop modeling tool can facilitates the BPEL modeling greatly with a GUI tool; virtual partner generator generates and deploys pseudo web services automatically at simulation period so that the developer need not invoke actual partner's web services, and makes the BPEL debugging and simulation easier; inspector generator generates and deploys automatically corresponding inspector web service which can inspect and record the any detail information of the process according to the developer's requirement; simulated BPEL generator can generate the simulated business process BPEL file according to the result of the virtual partner generator and inspector generator. This simulated BPEL file can be put into BPEL engine to test the business process; the result analyzer can replay the flow execution history record and display the specific information at running time.

We developed a prototype of WSCE as an Eclipse plug-in using Tomcat container. The environment has two states: modeling state and simulation state. In the following section, we use the example " simple" , which is used in BPEL engine of IBM alphaWorks, to explain and demonstrate how our environment works.

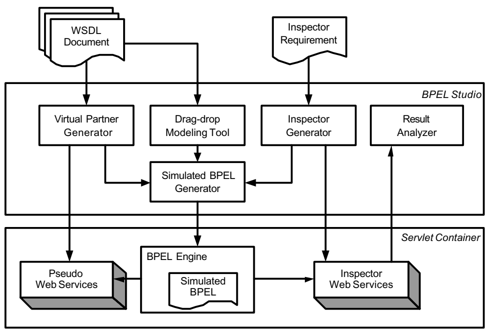
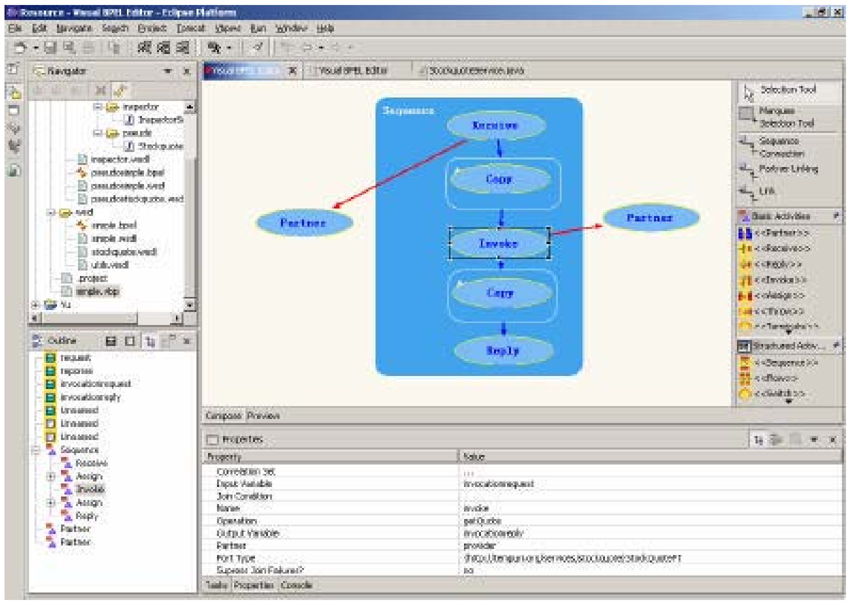


Fig. 1 The architecture of the BPEL environment

4.2 Drag-drop Modeling Tool

Visual representations are able to provide a high-level yet precise language which allows developers to express about concepts at their natural level of abstraction. So our drag-drop modeling tool provides the developers GUI tool in modeling state, and the developers can draw their business process easily, and input the value of the properties of the activities, then this tool can generate the BPEL file for them. To facilitate the developers inputting the properties of activities, the tool also parse the WSDL file of partner's web service, so that the developer can select naturally the values of properties from the menu provided by the tool. Our environment is based on Eclipse platform, the drag-drop modeling tool is developed as an Eclipse plug-in used GEF (Graphical Editor Framework)[3] package.

Fig. 2 shows the drag-drop GUI for "simple" example.



Property	Value
Correlation Set	...
Input Variable	invocationrequest
Join Condition	
Name	invoke
Operation	getQuote
Output Variable	invocationreply
Partner	provider
Port Type	[illegible]
Suppress Join Failure?	no

Fig. 2 The GUI of the modeling tool in the WSCE

4.3 Virtual Partner/Pseudo Web Service and Its Generator

When a developer finishes the BPEL file, he can not suppose that his BPEL file can achieve business goal unless that he run it in real business situation. However, many business partners can not provide test case. In some case, the business partner even needs to charge any transaction. If the developer sets up a simulated test environment by himself, it will cost him a lot of time. Considering that many simulated test environments of business process have a lot of similarity between each other, the developing environment can provide this kind of tool which is similar to code generator in [4] in simulation state for developers to facilitate development.

In our environment, we call this simulated web service as pseudo web service, and the service provider as virtual partner. Pseudo web service generator is responsible to generate and deploy them. In the real business situation, the back end system of partner's web service may be EJB (Enterprise JavaBean), CORBA(Common Object Request Broker Architecture), COM(Component Object Model), etc.

For simplification, here we use Java servlet to implement the back end system. The virtual partner needs to provide the same interface, but different binding message with the actual partner in their WSDL. Fig. 3 shows the workflow of pseudo web services generator.

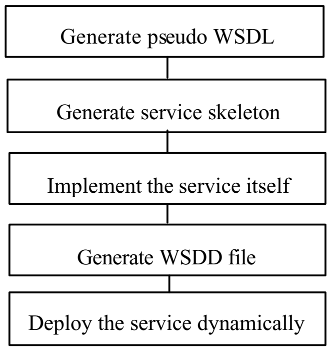

Generate pseudo WSDL
Generate service skeleton
Implement the service itself
Generate WSDD file
Deploy the service dynamically

Fig. 3 The workflow of the virtual partner generator

The detail workflow is

(1) When generating pseudo WSDL, the generator parses the partners' WSDL file, gets their binding information. The binding may be EJB binding, class binding (Java binding), or .net binding, etc. Here we uniformly map it to Java binding. Then the data type in XML schema is mapped into Java data type, and the partner web service URL is changed into local host. Then the pseudo WSDL file is generated.

(2) When generating service skeleton, we used Apache AXIS package, which can generate the service skeleton according to the pseudo WSDL file.

(3) Implement the service itself. The service includes its functionality and no-functionality. The non-functionality, such as the business performance, quality of service, etc should be concerned in the development cycle so that the developer can avoid any problems in the final runtime. For example, response time functionality is provided in our tool, That is, the service itself includes a random generator which is used to generate the runtime of the service according to users' requirement. The runtime information of the process and the pseudo web service are recorded into a file for analysis. The developer can use source code edit function to write the service functionality to test the business process. This is the only action that human being is involved in virtual partner generator, it's due to the consideration of the flexibility of service logic. Here we show a part service implementation.

```
public class StockquoteService
{
    private Logger m_logger= Logger.getInstance();
    private double from = 2.0;
    private double to = 2.5;
    private void logTime(int instanceID)
{
    GregorianCalendar now = new GregorianCalendar();
    m_logger.log(instanceID + "," + "this webservice
    begin at" + "," +now.get(Calendar.HOUR_OF_DAY)
        + "," + now.get(Calendar.MINUTE));
    int timeLength;
    timeLength = (int)(from + Math.random() * (to –
                        from));
    now.add(Calendar.MINUTE, timeLength);
    m_logger.log(instanceID + "," + "this webservice end
      at" + "," + now.get(Calendar.HOUR_OF_DAY) +
      "," + now.get(Calendar.MINUTE));
    }

    public java.lang.Float getQuote(java.lang.String
arg_0,java.lang.Integer instanceID)
{
    logTime(instanceID.intValue());
    // the service functionality
}
}
```

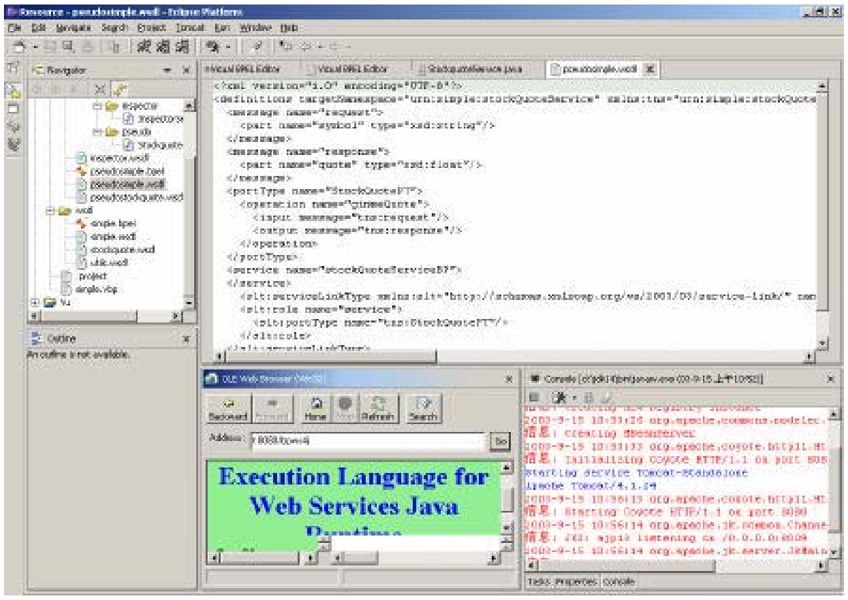
After the developer finishes the service code, the environment will compile it and the class file is generated correspondingly.

(4) Generate the service deployment file. The environment can generate the service deployment file and wrap all generated files into a *war* package and put it into corresponding directory.

(5) Dynamic deploy pseudo web services into a servlet container automatically. If the developer edits the service code himself, when he saves the service code, the other steps will execute automatically. Here we show the pseudo web service deployment interface. All the actions taken by the virtual partner generator are transparent to the developer; the only thing that the developer need to do is to open the simulation menu.

4.4 Process Inspector and Its Generator

Considering that BPEL specification is in developing process, we employ a third-party BPEL engine to run the business process. Thus the whole environment can be upgraded with the upgrade of BPEL engine. But we can not get the variables and runtime information in the BPEL engine unless it provides API for it. In order to monitor the simulated business process, and be adaptive to any BPEL engine in this situation, some inspectors should be added to the process. All the inspectors should not disturb the actual business process, and do the least affection on it, while the BPEL engine can accept and run it at the same time.



```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:simple:stockQuoteService" xmlns:tns="urn:simple:stockQuote
  <message name="request">
    <part name="symbol" type="xsd:string"/>
  </message>
  <message name="response">
    <part name="quote" type="xsd:float"/>
  </message>
  <portType name="StockQuotePT">
    <operation name="gimmeQuote">
      <input message="tns:request"/>
      <output message="tns:response"/>
    </operation>
  </portType>
  <service name="stockQuoteServiceBP">
  </service>
    <slt:serviceLinkType xmlns:slt="http://schemas.xmlsoap.org/ws/2003/03/service-link/" nam
    <slt:role name="service">
      <slt:portType name="tns:StockQuotePT"/>
    </slt:role>
```

```
2003-9-15 10:33:26 org.apache.commons.modeler.
信息: Creating MBeanServer
2003-9-15 10:33:33 org.apache.coyote.http11.Ht
信息: Initializing Coyote HTTP/1.1 on port 808
Starting service Tomcat-Standalone
Apache Tomcat/4.1.24
2003-9-15 10:56:13 org.apache.coyote.http11.Ht
信息: Starting Coyote HTTP/1.1 on port 8080
2003-9-15 10:56:14 org.apache.jk.common.Channe
信息: JK2: ajp13 listening on /0.0.0.0:8009
2003-9-15 10:56:14 org.apache.jk.server.JkMain
```

Fig. 4 One of the UI of simulation state

Therefore, the inspectors here are implemented as web services too. Inspector web services make our environment flexible and resilient. When the programmer switch to simulation state, he can input the requirement for the inspector, for example, if he can select " yes" in the property of " generate inspector" of " invoke" action in the GUI. All these inspector web services are generated by the component of inspector web service generator. And the process inspector generator does work as the following shown in Fig. 5, and details are also provided.

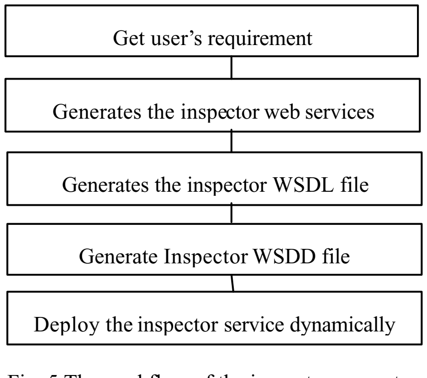

Get user's requirement
Generates the inspector web services
Generates the inspector WSDL file
Generate Inspector WSDD file
Deploy the inspector service dynamically

Fig. 5 The workflow of the inspector generator

(1) Gets user's requirement. The user can input his inspector requirement at any point in BPEL process. In the GUI tool, we provide an inspector attribute, if the user input " Yes" in this column, the requirement will be received by the system.

(2) Generates the inspector web services code and class. Because the inspector web service may need to recode the runtime information of the business process or partner's web services, monitor the variables of the business process, and provide the other users required information, etc, our tool provides source code edit function like pseudo web service generator, the developer can write the service functionality themselves according to their requirement. And the default function is to record the running time information. All these inspector web services are implemented using Java. After the code is generated, the JDK will compile it automatically and generate Java class file.

(3) Generates the inspector web services WSDL file. This file is generated by using Apache AXIS package automatically according to the service;

(4) Generates the inspector web services deploy description automatically, similar to the work of virtual partner generator;

(5) Dynamically deploys the inspector web services into appointed servlet container. When the developer finds that their requirements have changed, they can change the service functionality, and job (1)-(4) will adaptively change. New service will then be deployed dynamically. Same as the virtual partner generator, all the actions are transparent for the user.

At present, inspector web service in our environment can inspect the invocation of web services, such as " receive, invoke, reply" activities. Inspector web services can be extended to inspect any activities in BPEL process.

4.5 Simulated BPEL Generator

Simulated BPEL file is a little different from the original BPEL file, in which the inspector web services are part partners web services, and the BPEL process also calls the inspector web services. The simulated BPEL generator will parse the original BPEL file and insert the inspector web service invocation operations according to user's requirements, and generate the simulated BPEL file, which can be deployed into BPEL engine. So when the programmer switches to simulation state, he inputs the inspector requirements firstly and presses the *Generate* button. And the pseudo web service, inspector web service and simulated BPEL file are generated and deployed into local host container automatically. Finally, the programmer may deploy the simulated composite web service into the third party BPEL engine manually. Here we use the BPEL engine in IBM alphaWorks.

4.6 Result Analyzer

In order to test the business process written in BPEL, the programmer can write testing client by himself. If clients invoke the simulated composite web services, the pseudo web service and inspector web service will record the running time and other information. For example, if the recorded running information is like this:

*Instance ID, process *** beginning at t1;*
Instance ID, receive operation 1 at t1…

The result analyzer will analysis the above information and summarize the running information, for example, the number of testing instance, process average running time, partner's web service running time, et al. The analyzing gives developer the necessary information for monitoring and this help improve the design.

5. Experiments

In our environment, inspector is added to the business process. But if the inspector web service influences the process greatly, the result gotten from the simulation might leads to the serious problem for programmer to evaluate his whole design. In this part, we will try to explore the effects of inspector web service.

In our experiments, the example of "Simple" in BPEL engine of IBM alphaWorks is used, in which an atomic stock query web service is invocated. We deploy the processes without inspector and with inspector respectively to BPEL engine in a computer with AMD 1.8G CPU and 512M memory, and set the running time of pseudo web service as 2-2.5 seconds randomly. Twenty clients invoke the composite web services one by one and the every client invocating process time is measured.

Fig. 6 shows the whole process invocating time. In which the process without inspector and with inspector have little difference. The reason is that the running time of the atomic pseudo web service is randomly selected in a certain range, and their running time occupies the process running time greatly.

In order to evaluate the effect of the inspector web service, we subtract the running time of atomic pseudo web service from the running time of the process and compare the effect of inspector web service. Fig. 7 shows the time difference between the whole process and atomic pseudo web services, caused by the client invocation, server response, processing invocation, and inspector web service. The average time difference caused by inspector web service is 1597.75, nearly 10% of the process running time (not include the pseudo web service running time). We believe it is reasonable time for business process execution inspection, and the inspector web service has very little effect with the composite web service and our method is feasible.

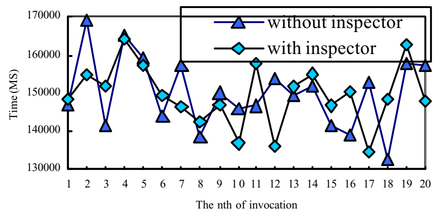


Fig. 6 The running time of simulated composite web service with and without inspector web service

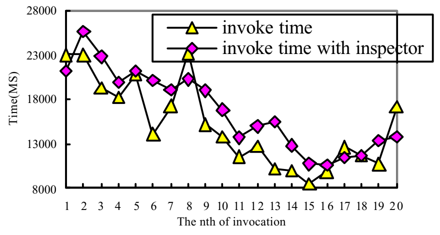


Fig. 7 The running time of simulated composite web service with and without inspector web service subtracting atomic web service

6. Related Work

So far, there are two main trends of web service composition: one is from industry [5] and another from Semantic Web community[5,6,7]. The industry views web services as abstract, standardized interface to business process, and has developed a number of XML-based standards to formalize the specification of web services, their flow composition and execution [5]. In particular, such Web languages as Universal Description, Discovery, and Integration (UDDI), Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP) are used to define standard ways for service discovery, description and invocation (message passing). Some web services need to communicate with each other, so that all of them can be composed into a business process, and the process is called web services orchestration or composition.

Early works in web service orchestration include WSFL from IBM, XLANG from Microsoft, WSCL, eCo from CommerceNet[8], SWORD[9], from Stanford, which uses its own simple description language and does not support any existing standards like WSDL or DAML-S. Now, BPEL4WS[1], joint effort from WSFL and XLANG, has superseded other language, and will be industrial de facto web services workflow specification. BPEL4WS, providing support for both executable and abstract business process, is essentially a layer on top of WSDL, with WSDL defining the specific operations allowed and BPEL4WS defining the operations can be sequenced. Although BEA contributed BPEL4WS, its current release of WebLogic can not support BPEL4WS well, so does current version of BizTalk of Microsoft. IBM provides an alpha release of BPEL4WS[10], and Collaxa Orchestration Server supports BPEL4WS with a GUI editor, an orchestration server and a management consol. Business process written in BPEL4WS can be debugged in some products, however, all of them can not provide pseudo web service concept to help simulation. Chandrasekaran[2] et al provide a simulation model in their project SCET to estimate performance, while their simulation is similar to the Entity Relationship model, and their simulation model is described by JSIM[11], a Java-Based Integrative Model Simulation and Analysis Environment, and need a JSIM Simulator to run it. Our simulation has two functionalities: debugging and performance estimation, which make the tool unique.

7. Conclusion and Future Work

In this paper, we propose the concept of IT level virtual partner and inspector in the web service composition. The virtual partner is a pseudo web service using the same interface with the actual partner but different binding message. The virtual partner can be invoked directly by the business process described by BPEL, so that the BPEL programmer can test both their application's functionality and non functionality performance early in the development cycle to avoid any problems in the final runtime. Due to involving non-functionality testing in virtual partner, the business man can select his partners according to the QoS requirement of the process. The inspector web services provide the users flexibility to adopt the third-party engines to inspect the process details. The IT virtual partners and inspector web service provide developers with a range of techniques to explore aspects of their programs.

Also, we implemented these concepts in a flexible Web Service Composition Environment (WSCE) for business process in this paper. The advantages of our environment include:

(1) Providing drag-drop UI programming model for developer to facilitate programming;

(2) Using the IT level concept of virtual partner and pseudo web service. This facilitates the developer to test the composed web services, automatically deploy testing environment and automatically change with the requirement. The method will bring easier and faster deployments, better continuity between development and operations, and a lower total cost of ownership.

(3) Being able to plug in any BPEL engine and the whole environment can upgrade with engine, which make its maintenance cost low.

At present, the Tomcat container is employed in the environment. Future work will involve Websphere Application Server container and provide more advanced UI to improve user's experience.

Reference

[1] Business Process Execution Language for Web Services Specification, version 1.1, avaible at : http://www-106.ibm. com/ deve loperworks/ webservices/library/ ws-bpel/

[2] Senthilanand Chandrasekaran, Gregory Silver, John A. Miller, Jorge Cardoso, and Amit P. Sheth, Web service technologies and their synergy with simulation, Proceedings of the 2002 Winter Simulation Conference, San Diego, California (December 2002) pp. 606-615

[3] GEF package, available at: www.eclipse.org/org/press release/may2002GEFpr.html

[4] Karthik Nagarajan, Herman Lam Stanley, and Y.W. Su. Integration of Business Event and Rule Management With the Web Services Model. Proceedings of the international conference on web service, Las Vegas, Nevada, USA. pp83-89, 2003

[5] Biplav Srivastava and Jana Koehler, Web service compo sition – current solutions and open problems, ICAPS 2003, Workshop on Planning for Web Services, 10, June, 2003, Trento, Italy

[6] Rubé n Lara, Holger Lausen, Sinuhé Arroyo, Jos de Bruijn, Dieter Fensel, Semantic Web Services: description requirements and current technologies, Semantic Web Services for Enterprise Application Integration and E-Commerce workshop, at the Fifth International Conference on Electronic Commerce (ICEC 2003), Pittsburgh, 1-3 October, 2003.

[7] Sheila A. Mcllraith, Tran Cao Son and Honglei Zeng, Semantic Web Services, IEEE Intellengent System, pp46-53

[8] Chris Peltz, web service orchestration, avaible at: www.hp .com/drc/technical_white_papers/WSOrch/SOrch estration.pdf

[9] Shankar R. Ponnekanti and Armando Fox, SWORD: A developer Toolkit for Web Service Composition, In Proceedings of The Eleventh World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii, May 2002

[10] BPWS4J, available at www.alphaworks.ibm.com/tech/bp ws4j

[11] JSIM, avaible at: http://nsr.bioeng.washington.edu/Soft ware/JSIM/