An automatic tool for deployment of BPEL processes in ODE Apache

Binh Nguyen Thanh* and Doan B. Hoang**

iNEXT – Centre for Innovation in IT Services and Applications
Faculty of Engineering and Information Technology, University of Technology, Sydney
* email: binhntf@it.uts.edu.au; ** email: dhoang@it.uts.edu.au

Abstract—Business Process Execution Language (BPEL) has become an important tool for composing new business processes from Web services and designing Grid workflows. However, running BPEL processes is often error prone procedure that involves a sequence of time consuming manual preparation. A deployment stage must be prepared to produce a specific deployment file, then all necessary files must be assembled to a specific deployment directory, and finally, a BPEL engine has to perform the compilation and error checking to produce runnable BPEL processes. This paper presents a tool for automating the deployment stage of runnable BPEL processes.

Keywords— BPEL, BPEL Engine, deployment tool, ODE;

1. Introduction

Recently, BPEL has become a widely accepted language for composition of new business processes from existing Web services. In turn, BPEL processes, which are also Web services, can be used in composing new business processes. The advantage of a BPEL process over a normal Web service is that a BPEL process allows the coordination of internal and external Web services according to its business process requirements. However, BPEL processes can not be deployed and run by Web services servers such as Apache Tomcat and Axis. They need specific BPEL engines which are usually built on top of Web services servers. Until now, there exist several BPEL engines such Active Endpoints' ActiveBPEL, Sun's Java Business Integration, and Apache's Orchestration Director Engine (ODE). Among them, the Apache ODE emerges as robust, easy-to-use and open source BPEL engine. In addition, this engine can be easily integrated in other BPEL environments. For instance, Eclipse BPEL Designer integrates ODE as its BPEL engine. The latest release of ODE 1.2 also support the latest BPEL specification 2.0.

In order to deploy a BPEL process in ODE, a sequence of steps has to be performed: Firstly, Partner Links have to be created from the WSDL files and from the BPEL process itself. Except one partner link that presents entities that will use the BPEL process, other partner links represent external web services that will be invoked in the BPEL process. After that, a specific deployment file (deploy.xml) has to be created according to the ODE’s structural specification of ODE. Lastly, all necessary files related to the ODE-BPEL process are copied to a specified directory to be compiled. If some errors occur, which is often the case, the log file of the Web server hosting the ODE has to be inspected manually for the error message and determine the reason for the error. Even if the compilation is successful, a runtime error can still exist and the inspecting and guessing process has to repeat again and again until successful.

It is clear from the above description that the pre-deployment process is complicated, time consuming and error prone. Even when one is familiar with the deployment of Web services, it is still an unnecessary complicated manual process that requires the understanding of BPEL, the intimate knowledge of WSDL structures and syntaxes, and the specific ODE's specific structure of deploy.xml files. It is also a tedious and time consuming process in manually creating the Partner Link component in various WDSL and deploy.xml files. The process is certainly prone to error. The most tedious part of the process is to find errors from the log file, which is extremely long and not well-structured as it contains all kinds of messages from the Web server, not just error messages relevant to one's BPEL deployment preparation process. This necessitates the development of an automated tool for the process.

In this paper, we present a tool that supports the deployment of BPEL processes in ODE. This tool will resolve the problems mentioned above by providing following capabilities:

- Automatically checking the validation of BPEL process file and related WSDL files. If they are all valid then a necessary deploy.xml file will be created.
- Automatically collecting all necessary files to deployment directory of ODE, and initiating...
the compilation process.
- Filtering and returning only relevant and useful information about the potential errors associated with the preparing and compilation process.
- Suggesting the possible solutions and reasons of the errors.
- Facilitating the integration with BPEL editors. In the next stage, the tool will be developed as an Eclipse plug-in, so that it can be easily plugged into our collaborative workflow editor that is currently being developed in our research centre.

2. Background

BPEL (Business Process Execution Language). Originally it was named BPEL for Web Services BPEL4WS, and has been changed to WS-BPEL. Recently, BPEL has become one of the most popular standard languages for Web services composition. There are two main approaches for service composition: orchestration and choreography. In orchestration composition, only one service as primary service directly invokes other services. Therefore, only this primary service knows the sequence of activities, states of requests and responses of invoked services. On the other hand, choreography composition does not involve a primary service. Interactions between services are supported by a coordination mechanism. BPEL only supports the orchestration composition.

In BPEL, this primary service is called the Business Process (BP), and other services are called Partners. There are two types of partner in BPEL: invoked partners that are invoked by the primary service and client partners that invoke the primary service. The potential relationship between a partner and a business process is called Partner Link. The interface between a process and a partner is through a portType which offers many operations.

only and input-output (request-response). Main components of a business process are depicted in figure 1.

BPEL engines.
ActiveBPEL engine[4]: the latest version 5.0 is an open source implementation of a BPEL engine, written in Java. It supports both BPEL4WS 1.1 and WS-BPEL 2.0 standards. One of the disadvantages of this engine is that the deployment of BPEL processes is not automatic and very complex in creating a deployment archive file.

BPEL engine in JBI (Java Business Integration): this engine is integrated in Netbeans environment only. One of its biggest limitations is that it only supports BPEL4WS 1.1 and not WS-BPEL 2.0.

ODE (Orchestration Director Engine): ODE is an open source BPEL engine of Apache. The latest version 1.2 has many interesting features such as:
- ODE supports for both the WS-BPEL 2.0 OASIS standard and the BPEL4WS 1.1.
- It supports two communication layers: Web Services http transport of Axis2 and ServiceMix on the JBI standard.
- It can be easily integrated with virtually any communication layer thank to high level API to the engine.
- It allows hot-deployment of processes: it means that you only need to copy all the necessary files to a specific directory (a deployment directory regulated by the engine), and then the running engine will automatically detect these files, compile them and make them ready for use.

3. Related work

Eclipse BPEL designer (EBD):
The current version 0.3 of EBD[3] has following features:
- It has a visual editor for editing BPEL files.
- It supports the creation of deploy.xml
- It also supports the deployment of all files in a BPEL engine.

However, this tool has a number of limitations:
- The visual editor can not present partner links, and lacks the external view of a process. It means that designer neither knows how the number of partners in the process and nor the number of links between the process and the
partners. This limitation makes the design of large and complex processes seemly impossible.

- Adding elements in the deploy.xml file is completely manual and time-consuming. It does not know when a partner has been added in the process file order to updates that information.
- The warning from the server and the engine after compilation is too long and detail, but lack of necessary information to help designer detect causes of the errors.

**Netbeans BPEL tools**
The latest version of Netbeans 6.5 integrates BPEL tools that contain a visual BPEL editor, WSDL editor and BPEL engine in JBI. However, it does not support ODE.

4. Design

**Design objectives**
As mentioned in the section 1, the outcome of the development of a BPEL process is a bulk of files which composes of a main BPEL file and related WSDL files (maybe there are XSD files associated with WSDL files, but on this stage, our tool does not take care of these files, because they do not affect the content of the deploy.xml file).

![Diagram](image)

*Figure 2: Main functionality of the tool*

The main task of our tool (shown in the figure 2) is to take this outcome as its input, then to check the validation of all the files, and finally to generate a deploy.xml file (in case all the input files are valid) or any error (otherwise) as its output. All the files are said to be valid when they satisfy the following conditions:

- The main BPEL process file exists and its content is valid. To check the validation of the content of this file, our tool only cares about the parts that declare and use Partner Links (it is the responsibility of the BPEL Engine to check and compile all content of the BPEL file).
- With each activity `receive`, `invoke` or `reply` using a Partner Link in the BPEL file, it requires a correspondent WSDL file representing the Partner Link. Therefore, this WSDL file must exist and contain the Partner Link. Moreover, it also has to contain the valid information about service name and port name.

**Functionalities of the tool**
The main steps of this task are shown in the diagram of figure 3.

![Diagram](image)

*Figure 3: Main steps of the tool*

- **Step 1:** Check Files. This step checks the completeness of necessary files. It means at least one BPEL file exists and each partner link in the BPEL file requires at least one WSDL file. Our tool only needs to have the directory that contains all the files. It will automatically detect the BPEL file and from this file to check all related WSDL files. If all necessary files exist, it goes to the next step. Otherwise, it stops and display errors.
- **Step 2:** Get all necessary information for `deploy.xml` file. The structure of `deploy.xml` file is presented in figure 4. As shown in figure 4, this step needs to find and retrieve the process name from BPEL file, and all information related to each partner link in the process such...
as service names and port names within the WSDL files. This step also helps check and find common errors such as missing partner links in BPEL file or in WSDL files, missing service and/or port in the WSDL files, etc. If all necessary information is valid then it moves on to the next step. Otherwise, it stops and shows the error detected.

- Step 3: Write all the information retrieved from step 2 into the deploy.xml file.

- Step 4: Finalize the deployment by copying all files to the deployment directory regulated by ODE engine. Normally, when all the files have already been found in this directory, ODE engine will detect and compile them automatically.

```xml
<?xml version="1.0" encoding="UTF-8"?>
xmlns:string="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:partnerLink="urn:partnerLink"
xmlns:partner="urn:partner"
xmlns:wsdl="urn:wsdl"
xmlns:bpel="urn:bpel"
xmlns:main="urn:main"
xmlns:partnerLink="urn:partnerLink"
xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<process name="main">
<active>true</active>
<partnerLink name="client">
<invoke partnerLink="Provider">
<service name="ws:mainService" port="mainPort"/>
</invoke>
</partnerLink>
</process>
</deploy>
```

Figure 4: Structure of the deploy.xml file

5. Implementation

Our tool has been developed in form of a JAVA program. In order to run the tool, it requires two arguments, one input is the directory that contains all necessary files such as BPEL and WSDL files; and one output is a deploy.xml file if all necessary files on the input directory are valid. Otherwise, it will display an error that helps find the reason why the deploy.xml cannot be produced. In this tool, in order to read and write BPEL and WSDL files that are all XML files, we use DOM (Document Object Model) API of JAVA. Actually, another API that can be used for this purpose is SAX (Simple API for XML). However, compared to SAX, using DOM allows us to modify the contents of files and is easier and quicker, because with DOM a file is read only once and then stored in the internal memory. Also, our tool does not require to read a file multiple times.

Our tool has been tested with several different projects of BPEL. With the projects that all files are valid, our tool also produces valid appropriate deploy.xml files that are ready for the deployment of these projects in deployment directory of ODE. In this case, our tool also supports two kinds of WSDL files:

- The normal WSDL file: with this type of files, the file contains all information about partner links, services and ports. Therefore, it is quite easy to get all this information by reading only one file.
- The wrap WSDL file: This type is an extension of a normal WSDL file. This file extracts the partner links from the appropriate normal WSDL file and wrap them in a new WSDL file. This wrap file has to import the normal file. Once the wrap WSDL file is created, the partner link information will be extracted, and then following the import link, the information about the service and port can also be extracted.

In the case, where the project files contain errors, our tool stops and displays the errors and the files containing the errors.

Figure 5: File structure of the example
An illustrative example

In the following section, we present a concrete example that demonstrates the use of our tool. In this example, we have developed a BPEL process that invokes an external web service from WeatherBug [5] to get the weather information on the globe. This example is developed by NetBeans version 6.1. Figure 5 shows the structure of the files of the example.

The main process is contained in the main.bpel file, which has two partner links: the client and the WeatherProvider. The workflow of this process is shown in Figure 6.

Among the two partner links, the client represents the one that uses this process (other processes, web services, etc.), and the WeatherProvider represents the external web service from WeatherBug. It can be seen that there are many operations in this web service, but in this example only one has been used, the operation called GetLiveCompactWeatherByCityCode() that gets online weather information of a city by its given code.

After the example has been developed, we can use our tool to deploy it in Apache Tomcat (we used version 5.5). Our tool needs two arguments, sourceDir, the directory that contains all source files, and destDir, the directory where the process is deployed (it usually is $CATALINA_HOME/webapps/odeWEB-INF/processes).

After running the tool, if no error is detected, a deploy.xml file will be created in the sourceDir, and then all files in the sourceDir will be copied to destDir, then the process is ready to be used. Figure 7 shows the structure of the files after running our tool. As shown, the deploy.xml file has been created and added.

Finally, we use the soapUI tool [6] to test the process. The result is shown in Figure 8. On the left window is the input of process which requires a city code. The right window shows the current weather information of that city.

6. Conclusions and future work

So far, our tool is only a standalone JAVA program. To make it more useful, in the next phrase of development, we will convert it to a plugin of Eclipse that can be integrated in our Collaborative Editor developed by Eclipse. New features also need to be added such as: support multi BPEL files, and more comprehensive set of errors.

This paper presents our work on the development of an automatic tool for deployment of BPEL processes in ODE engine. With our tool, the deploy.xml can be generated automatically from a main BPEL file and associated WSDL files. Moreover, our tool can check and find the errors that are invalid in the source files.
References