Journal of Computer Science and Cybernetics, V.28, N.3 (2012), 245–259

G-ODE, AN EXTENSION OF THE APACHE ODE FOR GRID SERVICES

BINH THANH NGUYEN[1], DUC HUU NGUYEN[1], THUY THANH NGUYEN[2]

[1]*Hanoi University of Science and Technology*
[2]*University of Engineering and Technoloty*

Tóm tắt. Gần đây, dịch vụ lưới và các hệ thống luồng công việc sử dụng BPEL đang thu hút nhiều sự quan tâm nghiên cứu trong môi trường tính toán lưới, bởi vì chúng cho phép các công việc hữu dụng có cấu trúc có thể được tạo dựng và được gọi một cách tự động trong môi trường lưới. Với các dịch vụ Web (Web services) thông thường (các dịch vụ không có trạng thái), việc tạo dựng và gọi chúng có thể được thực hiện bằng BPEL và các engine (là chương trình dịch và thi hành) của nó như ActiveBPEL và Apache ODE. Tuy nhiên, việc gọi các dịch vụ lưới (Grid services) (các dịch vụ có trạng thái) lại gây ra một vấn đề cho BPEL và các engine của nó, bởi vì chúng thiếu các đặc điểm hỗ trợ các dịch vụ có trạng thái. Trong một nghiên cứu trước đây của nhóm tác giả [1], vấn đề này đã được phân tích kỹ lưỡng, và từ đó một giải pháp đơn giản và ngắn gọn đã được đề xuất nhằm cho phép việc gọi các dịch vụ lưới từ Apache ODE engine. Bài báo nhằm tổng hợp các kết quả trước đây, cùng với kết quả cài đặt mới hoàn chỉnh cho giải pháp đã được đề xuất trên. Ý nghĩa thực tiễn của giải pháp đã đề xuất sẽ được minh họa rõ ràng hơn thông qua một ví dụ luồng công việc sử dụng BPEL. Cài đặt của chúng tôi, được gọi là G-ODE, là một sự mở rộng của Apache ODE, nhằm cho phép việc gọi dễ dàng cả hai loại dịch vụ là Web và Lưới.

Abstract. Grid services and workflows using BPEL have gained much interest in Grid computing recently as they allow useful structured tasks to be composed and invoked automatically within a Grid environment. With normal Web services (stateless ones), the composition and invocation can be done with BPEL and its engines such as ActiveBPEL and Apache ODE. However, the invocation of Grid services (stateful ones) presents a problem for BPEL and its engines because they lack features for supporting stateful services. In our previous work [1], this problem was deeply analysed and from that a clean solution was proposed allowing BPEL-ODE invocation of Grid services. This paper aims to combine the previous result with new complete implementation for the proposed solution. The meaning of our solution will be explained more clearly through a BPEL workflow example. Our implementation called G-ODE, is an extension of the ODE enabling both Web Services and Grid Services invocation.

Keywords. G-ODE; Grid services; BPEL; BPEL engine; ODE; Grid computing.

1. INTRODUCTION

Recently, Grid Computing plays an important role in resolving a real and specific problem of coordinated resource sharing and problem solving in dynamic, multi-institutional virtual

organizations [2]. Grid research has progressed to its third generation [3], which focuses on resolving problems that occur when large scale and autonomic grid systems need to be built. This generation has seen an increase in the adoption of service-oriented architecture and the development of a comprehensible architecture for large scale grid applications. The Open Grid Service Architecture (OGSA) was developed to support the creation, maintenance and application of ensembles of services in Virtual Organizations (VO). OGSA adopted the OASIS Web Service Resource Framework to bring Grid services closer to Web services community, allowing them to share and reuse tools that have been well developed for Web services.

In our previous paper [4], we proposed a grid collaborative framework that is both general purpose and plan supported. With the theoretical foundation based on the activity theory and designed on top of existing OGSA infrastructure, our proposed framework aims at accelerating the development of grid collaborative systems that consider working plans as central role. Our framework has a component called *Activity Planning* that is responsible for creating a new working plan or updating existing ones. The role of our plan is similar to a workflow in the workflow managament systems, which describe the activities and relationships between them before making the plan run. As suggested in [5], it seems to have so many workflow languages so far, that it is wiser to choose a suitable existing one, rather than to reinvent the wheels. Among the current workflow tools and languages that can support in creating working plans so far, the BPEL (Business Process Execution Language)[6] and BPMN (Business Process Model and Notation) [7] seem to be the best choices for our framework. With BPMN, abstract plans (similar to abstract BPEL process) will be proposed by many users for modeling real business processes. After that, these plans will be automatically transformed to BPEL processes acting as the working plans. The BPEL has been deployed successfully in composing workflows of Web services for many kinds of applications [8-10]. It is not surprising to see efforts to use BPEL for composing Grid services into higher level and structured tasks as Gridflows. However, due to the stateful nature of Grid services, BPEL and its engines cannot be deployed without additional features. The focus of this paper is to resolve this issue, while the other issue relating to automatic transformation techniques between BPEL and BPMN will be one of our next research directions.

Much effort in recent investigation attempts to overcome this problem. Some proposals have been suggested to invoke Grid services from BPEL [11, 12], but they are only for the ActiveBPEL engine[13]. The main issue with the ActiveBPEL is that it is not open source any more, therefore extension of this engine is not easy. For the open source BPEL engine, ODE (Orchestration Director Engine), we are not aware of any concrete solution so far. In our previous work [1], the issue of invocation of Grid services within the ODE engine was deeply analysed and from that a clean solution was proposed allowing BPEL-ODE invocation of Grid services. This paper aims to combine the previous result with new complete implementation for the proposed solution.

The structure of the paper is as follows. Section 2 presents some necessary background. Section 3 reviews some approaches in resolving the problem related to resource key in Grid services. The next section aims to investigate the current issue of ODE, and then the appropriate solution for this issue will be proposed in section 5. Section 6 describes the design and implementation of G-ODE. In section 7, some main related work will be reviewed. Finally, section 8 summarizes the contributions and concludes the paper.

2. BACKGROUND

2.1. Workflows for Grid

GSFL

GSFL (Grid Service Flow Language)[14] is one of the first workflow languages proposed for the Grid environment. Its development purpose is how to support the ability of composition of new workflows from existing Web services, and how to be compatible with the OGSA (Open Grid Service Architechture).

However, an important requirement of workflows for the Grid, the ability to invoke Grid services and integration of both Grid services and Web services, has not been mentioned in GSFL. Moreover, we have not seen any concrete implementations for GSFL, so it is very hard to evaluate the feasibility of this workflow language.

A-GWL

A-GWL (Abstract Grid Workflow Language) [15] is a high level workflow language for modelling workflows at abstract level. It means that its focus is mainly on description of the logic and flows of workflows, not on the their execution. This language bases on the activity diagram of UML (Unified Modelling Language), an popular object-oriented modelling language. This approach has an advantage of inheritance the widely-accepted modelling capability of UML. The workflows modeled by activity diagrams can be converted to A-GWL by a tool called Teuta [16].

However, similar to GSFL, this workflow language still lacks of concrete implementation and is without any support for invocation of Web service and Grid service.

GWEL

GWEL (Grid Workflow Execution Language) is the heart of the Grid Workflow Infrastructure proposed in [17]. Based on the OGSA, this infrastructure aims to leverage the concepts of the BPEL4WS. With OGSA, we can exploit the advanced Grid features such as instance factories, notifications and lifetime management. Leveraging BPEL4WS allows us to reuse the basic workflow functionalities, which are similar for Web service and Grid service. Our solution shares this idea, but we choose an another approach. Instead of creating a new workflow language as GWEL that is very complicated and spends much effort, we only expand the capability of one BPEL4WS engine (ODE engine) for enabling invocations of Grid services. Our approach is much more simpler and also more feasible.

2.2. WSRF

Before Grid services, a Web service refers to a stand-alone service with each instance is completely independent of any other instances of the same service as they do not keep any state information about themselves once they deliver their output to the requested Web client. This statelessness makes Web services client-server model simple, however, developing transactional services based on Web services requires complicated manipulation of the persistent states at the server end that is not consistent with the nature of stateless Web services. For this reason,

OASIS has adopted *Web Service Resource Framework* (*WSRF*), a standard that allows Web services to access their persistent states in a consistent and interoperable manner. In this framework, a state is called *stateful resources*. WSRF aims to model and manage stateful resources based on a construct called WS-Resource, which is composed of a Web service and its associated stateful resources [18]. WSRF defines means by which:

- WS-Resources can be created and removed.

- A stateful resource is used when message exchanges of Web service are executed.

- A stateful resource can be queried and modified via message exchanges of Web service.

Each stateful resource usually has many independent instances that may be created and destroyed. When a new instance of a stateful resource is created, normally by a Web service referred to as a *resource factory*, it may be assigned an *identity* (also called *resource key*).

WSRF defines a special kind of relationship, called *implied resource pattern*, between a Web service and its stateful resources. This relationship is a mechanism to associate a stateful resource with execution of message exchanges of a Web service. The term *implied* means that the stateful resource associated with a given message exchange is considered as an implicit input for the execution of the message request. *Implicit input* means that the stateful resource is not provided as an explicit parameter in the body of the message request. Therefore, the association occurs mostly in a dynamic manner, which is at the time of the execution of the message exchange [18].

To represent the address of a Web service deployed at a given network endpoint, WSRF uses the *Endpoint Reference* construct from WS-Addressing. The main part of an endpoint reference is an *Endpoint Address* of the Web service. The endpoint reference may also contain a metadata associated with the Web service such as service description information and *reference properties* (this name is used in WSRF version 1.1. In version 1.2 it has been changed to *reference parameters*). The reference properties play an important role in the *implied resource pattern*, as it is used to keep the *resource key* of the instance of stateful resource.

Grid services are the Web services that follow WSRF standard. They are also called WSRF-compatible Web services.

2.3. Axis2

Axis2 is the new version of Axis (Apache eXtensible Interaction System), a SOAP engine and a Web Service middleware tool. It is a SOAP messaging system with modular architectural design. The Axis2 Framework is built up of 7 core modules. Non-core/other modules are layered on top of these core modules [19, 20]. Among the seven core modules, SOAP Processing Module is relevant to our research. This module controls the execution order of the processing. Besides defining different built-in phases of the execution, the model supports extensible capability by permitting users to extend the Processing Model at specific plug-in places. The SOAP Processing Model is shown in figure **??**.

Two basic actions a SOAP processor are sending and receiving SOAP messages. To support these, the architecture provides two pipes (or messages processing flows) called ***In*** *Pipe* and ***Out*** *Pipe*. The implementation of these two pipes is via definition of two methods, send() and

receive() in the Axis2 Engine.

Extensible capability of the SOAP processing model is provided by handlers. When processing a SOAP message, only the handlers that are registered will be executed. Axis2 supports three scopes that the handlers can be registered in, global, service, or operation. The final handler chain will be calculated by combining the handlers from all the scopes.

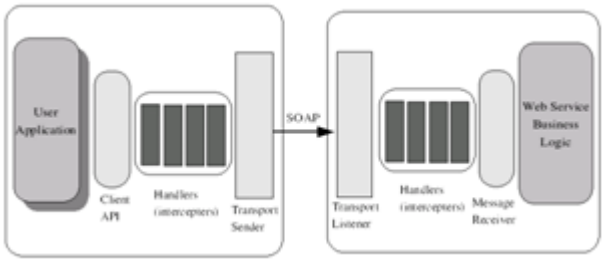

Figure 2.1. SOAP Processing Model of Axis2[19]

Acting as interceptors, the handlers process parts of the SOAP message and provide add-on services. The different stages of the pipes are called *phase*, which provides a mechanism to specify the ordering of handlers. A handler always runs inside a specific phase. Both Pipes have built-in phases, as well as the places for 'User Phases' which can be defined by users.

2.4. Apache ODE engine

ODE is an open source BPEL engine of Apache. The latest stable version 1.3 offers many interesting features such as:

- ODE supports for both the WS-BPEL 2.0 OASIS standard and the BPEL4WS 1.1.
- ODE supports two communication layers: Web Services http transport of Axis2 and ServiceMix on the JBI standard.
- ODE can be easily integrated with virtually any communication layer due to the high level API to the engine.
- ODE allows hot-deployment of processes. This means that one only needs to copy all the necessary files to a specific directory (a deployment directory regulated by the engine), and the running engine will automatically detect these files, compile them and prepare them ready for use.

The current ODE does not recognize all necessary information sent from a BPEL process and is unable to support the invocation of Grid services. Our solution aims to resolve this issue.

3. APPROACHES

A Grid service is a stateful web service: a Web service plus persistent *resources*. Each resource has its own ID called *Resource Key*. The combination of grid service address and its resource defines an *endpoint reference* (EPR). Normally, the value of grid service address is static and known before its running time. However, the value of resource key is dynamic and this presents a problem for the ODE to deal with Grid services. Possible approaches for resolving this problem include:

- Extending BPEL by adding new activities [12] into the standard BPEL. Among these activities, one (called *gridCreateResourceInvoke*) is used to get EPR, and other one (called *GridInvoke*) is for calling operations of the Grid service.
- Adding an operation in the Grid service that needs to be invoked, as suggested in [11, 21]. This operation will return EPR of an instance of that Grid service (therefore the operation is normally called *CreateResource*).

Clearly, extending BPEL is not a simple task, as it requires numerous changes in the standard BPEL engine, from the compilation of BPEL processes to the implementation of new activities.

Our effort focuses on the adding the operation to Grid services as it potentially provides cleaner and better solutions.

4. INVESTIGATION OF ODE'S PROBLEM

As mentioned above, the current ODE engine can only support the invocation of Web services and not Grid services. Before providing a concrete solution, this problem needs to be investigated in detail. Therefore, we developed a test aiming to determine how ODE engine invokes Web services and to locate exactly the problem when the ODE has to deal with invocation of an external grid service from a BPEL process.

Our test comprises of three parts:

- A grid service called MathService, which has been adapted from the example in [22] (The example can only run in Globus Toolkit 4.0. Necessary changes have been done to make it run in GT 4.2). This grid service already follows the factory/instance pattern [21]. This pattern requires each grid service having its factory service called MathFactoryService, a normal Web service whose main role is to create instances of the Grid service. Besides creating new instances, this operation (called *createResource* in our example) also returns *ResourceKeys* for the instances. The Globus Toolkit version 4.2 [23] has been used to deploy this grid service in its web service container.
- A BPEL process called Test-FactoryGS-V2: Diagram of the process is illustrated in Figure **??**. More details of this process will be explained later.

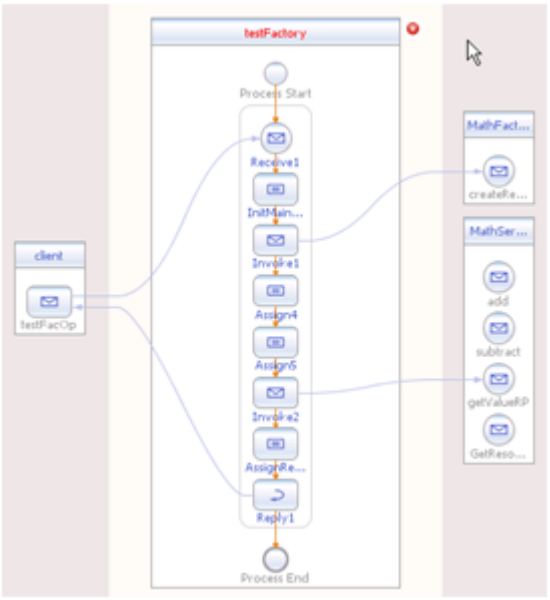

Figure 4.2. Diagram of BPEL process Test-FactoryGS-V2

- ODE engine and its deployment in Apache Tomcat 6.

In order to locate the problem, our BPEL process is designed to inspect three activities:

- The first activity is to invoke the Grid service in order to get its *ResourceKeys*.
- The second activity is to assign this *ResourceKeys* to a *PartnerLink*, allowing the PartnerLink to carry the *ResourceKeys*.
- The third activity is to use the *PartnerLink* to invoke another operation of the Grid service.

Main activities of the BPEL process Test-FactoryGS-V2

In order to meet our goal, the designed BPEL process will perform three activities as follows:

(1) Invoking the operation *createResource* of *MathFactoryService* and store the returned *ResourceKey* in a variable called *CreateResourceOut* by the following code:

```
<invoke name="Invoke1" partnerLink="MathFactoryServicePL"
operation="createResource"
    xmlns:tns="http://www.globus.org/namespaces/examples/core/FactoryService"
    portType="tns:FactoryPortType"
    inputVariable="CreateResourceIn"
outputVariable="CreateResourceOut"/>
```

(2) Assigning the variable *CreateResourceOut* to a *PartnerLink* by the following code:

```
<assign name="Assign5">
<copy>
    <from variable="CreateResourceOut" part="response" />
    <to partnerLink="MathServicePL"/>
    </copy>
</assign>
```

```
<invoke name="Invoke2" partnerLink="MathServicePL"
  operation="getValueRP"
  xmlns:porttype="http://www.globus.org/namespaces/examples/core/MathService_instance"
  portType="porttype:MathPortType"
  inputVariable="GetValueRPIn"
  outputVariable="GetValueRPOut"/>
```

(3) To invoke one operation of the grid service *MathService* by the following code:

Results of the test

When (1) has been invoked successfully, the returned value *CreateResourceOut* contains the *ResourceKey*, as shown by bold part in the returned message:

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  ...
  <soapenv:Header>
  ...
  </soapenv:Header>
  <soapenv:Body>
    <createResourceResponse xmlns="http://www.globus.org/namespaces/examples/core/
    FactoryService">
      <wsa:EndpointReference>
        <wsa:Address>http://127.0.0.1:8082/wsrf/services/examples/core/factory/MathService
        </wsa:Address>
          <wsa:ReferenceParameters>
            <ns1:MathResourceKey xmlns:ns1="http://www.globus.org/namespaces
            /examples/core/MathService_instance">12692431
            </ns1:MathResourceKey>
          </wsa:ReferenceParameters>
        </wsa:EndpointReference>
    </createResourceResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The assignment of *CreateResourceOut* to the PartnerLink *MathServicePL* was performed successfully. However, the invocation of (3) failed. Through checking the in/out messages, the exact cause of the problem was found. In the message sent to invoke operation *getValueRP* of the grid service, the necessary *MathResourceKey* was not found.

In ODE, two steps are required for getting *endpoint reference* (EPR). The first step is

to invoke the operation *CreateResource* that returns the EPR from the Grid service, and then the returned EPR needs to be stored in a *PartnerLink* for later use. Surprisingly, even though the EPR contains the *ResourceKey*, the ODE still supports these two steps very well (please remember that in a Web service, its EPR contains no *ResourceKey*, and the ODE only supports Web services, not Grid services).

The issue of ODE in the Grid service invocation occurs exactly when the *PartnerLink* carrying the EPR is used to invoke operations in the Grid service. The reason is that the message used in this invocation lacks the *ResourceKey*.

5. SOLUTION TO THE PROBLEM

It is clear from our investigation that in order to enable the ODE engine to support Grid services invocation, we have to develop a mechanism to allow adding suitable ResourceKey into the message sent to invoke an operation of the Grid service. Because ODE engine is built on Axis2, which uses handlers (interceptors) to proceed messages, we need to develop new handlers whose roles are to find out necessary ResourceKeys and add them into the suitable messages. Furthermore, these handlers have to be positioned in proper positions in the series of existing handlers. In ODE, each handler is implemented by a Java class.

Inherited from Axis2, the ODE engine also uses the file axis2.xml to setup its configuration parameters such as TransportSender, TransportReceiver, MessageReceiver, Handlers and Handlers Processing Order, etc. Following is the phaseOrder part, which setups the Handlers Processing Order, extracted from configuration file axis2.xml of the ODE engine. In this extraction, the bold part shows our handlers that would be inserted after they have been implemented.

6. G-ODE

6.1. Design

Requirements for Grid Services

Our solution requires that each Grid Service needs to have a special operation called *createResource* which aims to create a *resource key* that will be used by following invocations of other operations of the Grid Service. Calling this operation only uses normal invocation like calling operations of Web Services. The *createResource* returns an EndpointReference including the *resource key* of the Grid Service. This *resource key* will need to be inserted properly in messages sent to invoke other operations of the Grid Service.

Message handlers

Following the processing model of ODE (inherited from the Axis2), our implementation composes of two handlers:

```
<phaseOrder type="InFlow">
<!– System pre defined phases –>
   <phase name="ProcessHeader">
     <handler name="ResourceKeyInHandler"
       class="our class here">
       <order phase="PostDispatch"/>
     </handler>
   </phase>
</phaseOrder>
<phaseOrder type="OutFlow">
<!– user can add his own phases to this area –>
   <phase name="ProcessHeader">
     <handler name="ResourceKeyOutHandler"
       class="our class here">
       <order phase="PostDispatch"/>
     </handler>
   </phase>
   ...
</phaseOrder>
```

- **GridHandlerIn**: this handler included in the phaseOrder, InFlow is responsible for detecting the *resource key* from the message returned by the *createResource* operation of the grid service. If the *resource key* is found, it will be saved to be used later in invocations of other operations of the grid service.
- **GridHandlerOut** :this handler included in the phaseOrder, OutFlow is responsible for searching a proper resource key among the saved ones, and inserting it into the header of the message sent to invoke an another operation of the grid service.

One problem has been detected when looking for the way to store the resoure keys, due to the fact that multi instances of a grid service can be created and run in concurrence. Because in ODE, each instance of a grid service has an unique ConfigureContext, therefore we use it to store resource keys.

Figure **??** shows the sequence diagram that describes the processing sequence of these two handlers. It consists of the following steps:

Step 1: a BPEL process invokes the MethodA of a Grid Service (MethodA may be createResource or other one). When receiving this invocation, ODE engine will create an appropriate SOAP message that will be sent to GridHandlerOut to process.

Step 2: At GridHandlerOut, it checks the existence of resource key in the ConfigureContext. If it does not exist, the MethodA must be createResource, and GridHandlerOut allows the message pass without any modification. Otherwise, if resource key exists, then GridHandlerOut will insert it to the message before sending.

Step 3: The operation MethodA of the grid service is actually called, and it returns a soap message that will be processed by GridHandlerIn.

Step 4: The content of the message will be checked to look for the EndpointReference part. If this part exists, it means the operation called is createResource. Therefore, the message will contain the resource key that will be extracted and saved in the ConfigureContext.

6.2. Implementation

In ODE, each handler is a class inheriting from the class **AbstractHandler.**

After implementation of the two handlers GridHandlerIn and GridHandlerOut, they need to be added in the configuration file *Axis2.xml* as shown in Table 1.

6.3. Experiment results

Our test aims to check how well many different grid services can be called from a BPEL process through our G-ODE. The focus of our test here is not on the performance of the BPEL process running on the G-ODE, but only on the invocation capability of grid services from the BPEL process. Therefore, a simple BPEL process with two invocations has been choosen. The more complex BPEL processes aiming to test the performance of the G-ODE will be composed in our future work. In this test, only a simple BPEL process which calls two different grid services MathService and CalcService, has been built and run (see Figure 7.4).

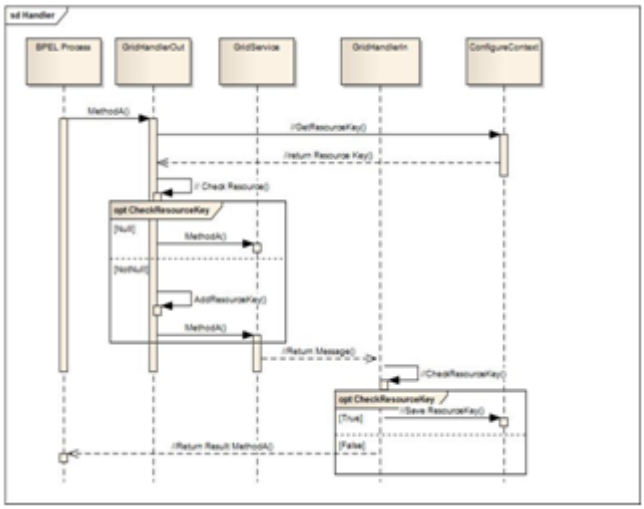

Figure 6.3. Sequence diagram of the two handlers

This process does as follows:

- Firstly, two parameters X and Y are declared. Then two instances of these two grid services are created (by activities CreateCacl and CreateMath). This aims to retrieve and save the resource keys of these instances.

Table 1. Configuration file Axis2.xml after adding two new handlers

```
<phaseOrder type="InFlow">
   <phase name="ProcessHeader">
     <handler name="GridHandlerIn"
       class="org.hust.hpcc.grid.GridHandlerIn">
       <order phase="PostDispatch"/>
     </handler>
   </phase>
</phaseOrder>
<phaseOrder type="OutFlow">
<!– user can add his own phases to this area –>
   <phase name="GridHandlerOut">
     <handler name="ResourceKeyOutHandler"
       class="org.hust.hpcc.grid.GridHandlerOut">
       <order phase="PostDispatch"/>
     </handler>
   </phase>
   ...
</phaseOrder>
```

- Secondly, two parallel flows for invocations of these two grid services are created. In the first flow, it calls Add(X) operation, and then Multiply(Y) operation of the CalcService (it means the result would be X*Y, because the initial state of the service is zero). In the second flow, it only calls Add(X) operation of the MathService (the result would be X).

- Finally, output with two above results will be returned.

  Figure 7.5 shows the result running this BPEL process. As can be seen from Figure 7.5, using G-ODE, two different grid services can still be invoked easily. From the BPEL process developers' point of view, calling grid services is nearly the same as calling web services. They even do not have to care about the existence of the resource keys of grid services.

7. DISCUSSION ON THE SOLUTION

We are not aware of any existing solutions that enable the invocation of Grid services with ODE engine, thus we can only comment on solutions that use different BPEL engines.

i) Compared to the solution in [12] that extends the standard BPEL by adding new activities, our solution is much simpler on two points. First, with the solution that extends BPEL, the users will not only have to learn newly added activities, but also have to determine whether the service is a Grid service or a Web service when they invoke the service. With our solution invoking a Grid service is just like invoking a normal Web service. Second, the solution that extends BPEL requires many changes, from the module for compilation of BPEL process, to the runtime module for the compiled process. In contrast, our solution does not make any

change to these modules. It only adds some independent handlers (each one is a Java class), and makes suitable changes only in the configuration file to register the new handlers.

ii) The solution proposed in [11] is similar to ours but for the ActiveBPEL engine, not the ODE engine. Similar to the ODE, this BPEL engine has not supported Grid services invocation, but only Web services invocation. This solution, however, has not mentioned this issue in the ActiveBPEL engine. Only using tricks in a BPEL process and without overcoming the issue of the BPEL engine, it seems very hard to invoke successfully Grid services from the process.

In brief, our solution is the only one available so far for the ODE BPEL engine for invoking Grid services. It does not require modifications of the BPEL engine, and requires little effort from BPEL process developers.

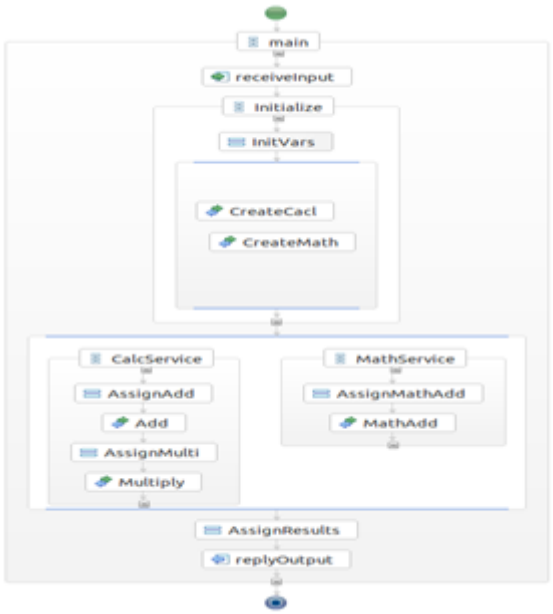

Figure 7.4. Our testing BPEL process

8. CONCLUSIONS

The contribution of the paper is summarized as follows. It resolves the problem with the ODE BPEL engine when invoking stateful Grid services. It offered a simple and practical solution that is not known to exist.

With our solution, the integration of both Web services and Grid services in BPEL processes not only becomes true, but also the application of the integration is quite easy. The significance of the integration is that it allows various Grid services to be composed into structured Grid/Workflows using BPEL and to be invoked through the Grid infrastructure underneath. This implies complex sequences of tasks can be composed and automated over the Grid computing environment rather than simple Grid requests.

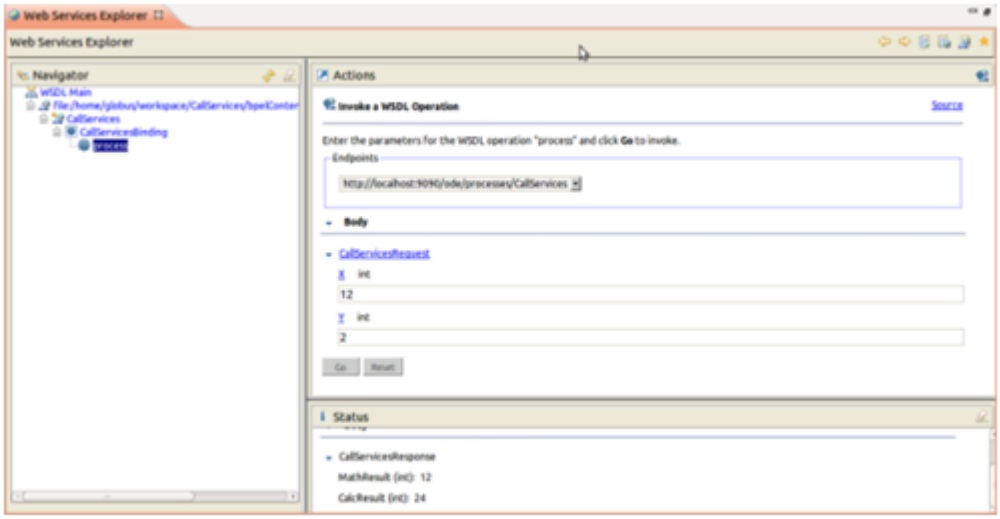

Figure 7.5. The results of running the BPEL process with X=12 and Y=2

However, due to the facts that BPEL is still a low level workflow language that requires many error prone and complicated little things such as initilization of variables, data conversion. Therefore it is not easy, time consuming for unexperienced users to compose BPEL processes. Moreover, BPEL still lacks of some advanced features of the Grid such as notification mechanisms and lifetime management of the invoked services. Overcoming these limitations of the current BPEL is our important research directions in the future.

REFERENCES

[1] Binh Thanh Nguyen, Doan B. Hoang, and T. T. Nguyen, Enabling grid services from BPEL process using ODE engine, *ICCSIT 2011,* Chengdu, China, 2011.

[2] C. K. Ian Foster , Steven Tuecke, The anatomy of the grid, *International Journal of High Performance Computing Applications* (2001).

[3] M. A. B. David De Roure, Nicholas R. Jennings, Nigel R. Shadbolt, *The Evolution of the Grid*, John Wiley & Sons, 2003.

[4] Binh Thanh Nguyen, D. B. Hoang, Building a plan-supported grid collaborative framework, *2nd International Cenference on Communications and Electronics (ICCE 2008),* Golden Sand Resort, Hoi An City, Vietnam, 2008.

[5] W. Tan, P. Missier, R. Madduri, I. Foster, *Building scientific workflow with Taverna and BPEL: A comparative study in caGrid*, 2009.

[6] OASIS, *Web services business process execution language version 2.0*, OASIS Standard, OASIS, 2007.

[7] O. M. Group, *Business Process Model and Notation (BPMN) Version 2.0*, Object Management Group, 2009.

[8] I. Morrison, B. Lewis, S. Nugrahanto, Modelling in clinical practice with web services and BPEL, *IGI Global* (2006) 45–57.

[9] W. van der Aalst, J. Jorgensen, K. Lassen, R. Meersman, Z. Tari, *Let's go all the way: from requirements via colored workflow nets to a BPEL implementation of a new bank system*, Springer Berlin/Heidelberg **3760** (2005) 22–39.

[10] M. Held, *Collaborative BPEL design with a rich internet application*, 2008.

[11] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, M. Robinson, *Grid service composition in BPEL for scientific applications*, Spinger-Verlag, 2007.

[12] T. Dornemann , Thomas Friese , S. Herdt, B. Freisleben, *Grid workflow modelling using grid-specific BPEL extensions*, German e-Science, 2007.

[13] ActiveBPEL engine at http://www.activevos.com/community-open-source.php.

[14] S. Krishnan, P. Wagstrom, G. V. Laszewski, GSFL: A workflow framework for grid services, Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439, 2002.

[15] T. Fahringer, S. Pllana, A. Villazon, M. Bubak, G. van Albada, P. Sloot, J. Dongarra, A-GWL: abstract grid workflow language, *Computational Science - ICCS 2004* **3038** (C. S.-I. 2004) (Ed.: Springer Berlin / Heidelberg 2004) 42–49.

[16] Sabri Pllana, Jun Qin, T. Fahringer, Teuta: A Tool for UML based composition of scientific grid workflows, *1st Austrian Grid Symposium*, Schloss Hagenberg, Austria (2005).

[17] D. Cybok, A grid workflow infrastructure, *Concurrency and Computation: Practice and Experience* **18** (2006) 1243–1254.

[18] Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Don Ferguson, Frank Leymann, Martin Nally, Igor Sedukhin, David Snelling, Tony Storey, William Vambenepe, S. Weerawarana, *Modeling stateful resources with web services version 1.1*, 2004.

[19] Apache Axis2 architecture guide at http://ws.apache.org/axis2/1_5/Axis2ArchitectureGuide.html.

[20] Axis2 configuration guide at http://ws.apache.org/axis2/1_5/axis2config.html.

[21] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, M. Robinson, Orchestrating WSRF-based grid services, *School of Computing and Information Sciences Florida International University* (2007).

[22] B. Sotomayor, *The globus toolkit 4 programmer's tutorial*, 2005.

[23] Globus toolkit 4 at http://www.globus.org/toolkit/.

Received on July 11, 2012

Revised on December 18, 2012