Analysis of Grid Service Composition with BPEL4WS

Kuo-Ming Chao[1], Muhammad Younas[1], Nathan Griffiths[2], Irfan Awan[3], Rachid Anane[1], C-F Tsai[4]

[1]*School of MIS, Coventry University, UK*
{k.chao, m.younas, r.anane}@coventry.ac.uk
[2]*Department of Computer Science, University of Warwick, UK*
nathan@dcs.warwick.ac.uk
[3]*Department of Computer Science, University of Bradford, UK*
i.awan@scm.brad.ac.uk
[4]*Department of Industry Management, Aletheia University, Taiwan*
tsai@email.au.edu.tw

Abstract

The Open Grid Services Infrastructure (OGSI) defines a distributed system framework by integrating Grid and Web services technologies to facilitate resource sharing. In OGSI, Web services are supplemented with additional features in order to meet the requirements of Grid computing. However, the issue of Grid service composition is not well addressed in the OGSI framework. We apply BPEL4WS (Business Process Execution Language for Web Services) as a business workflow description language for the composition of Grid services. We provide an in depth analysis of BPEL4WS and OGSI in terms of their similarities and differences in areas such as life cycle management, Web service instantiation and instance group management. Based on our analysis we propose a high-level architecture to compliment OGSI with BPEL4WS for defining process workflow among Grid services. We describe a prototype system which shows how the proposed architecture can be used in modelling or orchestrating Grid services with BPEL4WS.

1. Introduction

Web services are becoming an increasingly popular technology for Internet application development, receiving a significant investment of resources from both industrial and academic communities [1,2]. They provide a new solution to enable business interactions dynamically over the Internet, addressing issues such as application-to-application communication and system interoperability. The main advantage of Web services is that they allow applications to be loosely coupled, in contrast to traditional distributed systems that tend to be tightly coupled. Web services are based on standard technologies and protocols including SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), UDDI (Universal Discovery, Description, and Integration), and XML (Extensible Markup Language) [3, 4,5]. A Web service is an independent entity that can be advertised over the Internet. Users and developers can orchestrate multiple Web services to form useful and complex services in order to meet their particular requirements.

In the light of the maturity and popularity of Web service technologies, OGSI [6] adopted Web services as a new standard for developing an open Grid technology. The benefit of Web services is that the developers of Grid applications can benefit from broader support (both commercial and otherwise) for Web services standards. OGSA (Open Grid Services Architecture) not only introduces Web services to Grid technology, but also attempts to integrate Web service technologies with existing Grid standards by adding extra features such as instance management, notification mechanisms, and stateful interaction between Grid resources. This results in the introduction of the notion of *Grid service*. OGSA provides a platform GT3 (Globus Toolkit 3) [7] that allows a higher-level mechanism or language to incorporate Grid services into applications. GT3 is an implementation of OGSI.

This paper investigates how BPEL4WS [8] can be utilised in the composition and execution of Grid services. BPEL4WS is a formal description language for defining the workflow between Web services. It provides a standard process integration model to manage complex interactions between Web services. It also supports sequences of peer-to-peer synchronous and asynchronous message exchanges within stateful and long running interactions involving multiple parties. BPEL4WS engines, such as BPWS4J and Collaxa, provide a runtime environment for the composition and execution of Web services.

We propose a high-level architecture wherein BPEL4WS is used for defining the process workflow among Grid services. We develop a prototype system in order to demonstrate the proposed architecture in modelling and orchestrating Grid services with BPEL4WS. In this paper, we do not attempt to address the issues involved in achieving a full seamless integration of BPEL4WS and OGSI, but rather we demonstrate the feasibility of applying these two technologies together, and we identify the fundamental problems regarding their interoperability.

The remainder of this paper is structured as follows. Section 2 describes the related technologies; in particular Web services, Grid services in the OGSA, and the characteristics of BPEL4WS. The proposed architecture to enable BPEL4WS to compose grid services, along with a simple prototype, is described in Section 3. In Section 4, we discuss a number of issues regarding the integration of BPEL4WS and Grid services. Finally, Section 5 draws conclusions about applying BPEL4WS to Grid services.

2. Related Technologies

This section provides the description of Web services, Grid services, and BPEL4WS.

2.1 Web Services

A Web service is a service that contains a collection of operations to enable its interaction with the environment over the Internet through standardised XML messaging [9, 10,11]. The Web services platform is built on existing and emerging standards and technologies such as HTTP, XML, SOAP, WSDL, and UDDI. These technologies and standards are organized into four layers, comprising network, messaging, service description, and service publication and service discovery layers.

The lowest layer of the Web services framework is the network layer. Web services that are publicly available on the Internet use commonly deployed network protocols such as TCP/IP, HTTP, FTP, and IIOP.

In Web services, messages are communicated between participating systems using the XML-based SOAP protocol. SOAP provides an enveloping mechanism so as to communicate document-based messages.

WSDL facilitates the process of service description. Each service provider uses WSDL in order to describe the details of the services it provides. Services are defined through WSDL as collections of network endpoints, or ports [3]. In order to define services, a WSDL document contains several distinct elements, including: *portType* (an abstract description of the port); *message* (a typed definition of the data); *operation* (describes an action which is supported by the respective service); *port* (specifies an address for a binding); and *service* (aggregates a set of related ports).

The advantage of Web services is to provide a platform to allow business applications to interact or interoperate in a heterogeneous environment. However, use of Web services assumes that business applications have relatively simple interactions and only require a stateless model. This is inadequate to support complex applications demanding complex interactions, and long-lived stateful interactions. OGSI attempts to address some of these issues by incorporating a number of features introduced below.

2.2 Grid Services

The OGSI specification, utilises the WSDL and XML schema definition languages from Web services to define an extended component model [6]. The aim of the specification is to address the common issues that occur in sophisticated distributed applications, such as the management of distributed long-lived states. In order to achieve this aim, OGSI defines the notion of a *Grid service instance* [6]. "A Grid service instance is a (potentially transient) service that conforms to a set of conventions (expressed as WSDL interfaces, extensions, and behaviours) for such purposes as lifetime management, discovery of characteristics, notification, and so forth." [6]. The OGSI specification not only inherits the interoperability features from Web services, but also includes the following features.

- *Stateful interactions*: *serviceData* is the OGSI approach to stateful Web services. It exposes a service instance's state data to service requestors for queries, updates and change notifications [6]. The concept of *serviceData* is similar to a JavaBean. Thus, each item of data is associated with a set of methods (e.g., *get* and *set*) to access the state of data (attributes).
- *References*: OGSI uses Grid Service Handles (GSH) to name and manage Grid service instances. A client wishing to communicate with a service instance must map the GSH to a Grid Service Reference (GSR). This is because a GSH only contains a minimal set of information, such as a URI and it does not carry sufficient information to allow a client to communicate directly with the service instance. Instead, a GSR contains all information that a client requires to communicate with the service.
- *Collection of service instances:* OGSI allows a number of services to be grouped together so that they can be easily maintained by clients. A Grid service can define its relationship with other member services in the group. Services can join or leave a service group.

- *Life Cycle management*: This gives a client the ability to create and destroy a service instance according to its requirements.
- *Inheritance:* OGSI adopts some of the features from the WSDL 1.2 such as *portType* inheritance which allows one *portType* to extend from other *portTypes*. To distinguish between WSDL 1.1 and 1.2 [12], OGSI uses GWSDL to name the WSDL 1.2 *portTypes*.
- *Asynchronous notification*: OGSI provides a facility for asynchronous notification of state change using a pull/push mechanism.

In summary, the OGSI specification is an attempt to provide an environment for Grid services to be more manageable within large and complex distributed applications, and also to provide a platform for higher-level mechanisms to compose services.

2.3 BPEL4WS

BPEL4WS is an industry standard specification for defining the workflow between Web services [8]. It is intended to provide a workflow language to model complex and non-deterministic business processes. The characteristics of correlating business processes often depends on the data and BPEL4WS provides a set of activities to model data-dependent behaviours. BPEL4WS provides conditional and time-out constructs in order to address non-deterministic situations which often occur in business processes. BPEL4WS also provides developers with the ability to specify exception conditions and their consequences, including recovery sequences. The most important feature of BPEL4WS is to support business process coordination among multiple parties. This enables the outcome (success or failure) of units of work at various levels of granularity of the business processes. BPEL4WS enables modelling of long-running interactions between business processes with nested units of work between them and each with its own data requirements.

BPEL4WS is built upon three XML-based specifications: WSDL 1.1, XML Schema 1.0 and XPath 1.0. Partners are used by BPEL4WS to model interacting services in business processes. Each partner has a unique name and other services can interact with the partner through the name. Each partner is associated with a WSDL document, which describes the information that a service contains. The process model allows developers to specify the relationships between partners through a set of pre-defined activities in order to orchestrate Web services.

In BPEL4WS, the business process begins with a *receive* activity that receives a request from the client and triggers the process as a whole. The *reply* activity is the end of the process that responds to the request associated with a *receive* activity. The *invoke* activity allows invocation of an operator associated with *portTypes* (which is defined in a partner Web service). The state of messages related to business process is temporarily stored in variables.

Developers can handle known and unexpected exceptions with *throw* and *compensate* activities. The response to external events can be specified through event handlers. Control flow in BPEL4WS is similar to traditional structured process control containing constructs such as *while*, *switch*, and *sequence*. The *sequence* activity defines blocks that contain one or more activities that are performed sequentially. A *flow* activity allows the activities within the block to be performed concurrently. A *link* activity allows concurrently running activities to establish inter-dependency. Finally, the *correlation* construct specifies that only correlated instances can be invoked.

3. The proposed architecture and implementation

BPEL4WS was originally designed to orchestrate standard Web services, but it has not been used to orchestrate Grid services due the following issues. As described earlier, OGSI introduces GSH and GSR so as to reference Web service instances, but this is not supported by the BPEL4WS specification. Additionally, the adoption of certain WSDL 1.2 features for the Grid service interface descriptions, is not recognisable by BPEL4WS. This is because the current version of BPEL4WS is still based on WSDL 1.1.

In this paper, we propose an architecture (as shown in Figure 1) in order to alleviate the above problems and to enable Grid service composition via BPEL4WS. In the proposed architecture, we wrap Grid service clients as Web services called Proxy Web Services. All of the interfaces defined for the Grid services are re-defined in Java beans as an XML complex type (in WSDL) with a public Grid service instance attribute. An additional operator, *startGService*, is defined and implemented in the Proxy Web Service. This operator is to create new Grid service instances. The process of a series of activities being carried out is described as follows.

The BPEL4WS user initiates the client. The Proxy Web Services are invoked according to the workflow descriptions in BPEL4WS. The Proxy Web Services will trigger corresponding Grid services through an embedded *startGService* operator. The *startGService* operator is the standard procedure for creating a Grid service instance by calling a GSH, holding its returned value, and mapping it to a GSR. When a Grid service instance is created, the *startGService* operator obtains a reference and stores it in the predefined public Grid service instance attribute.

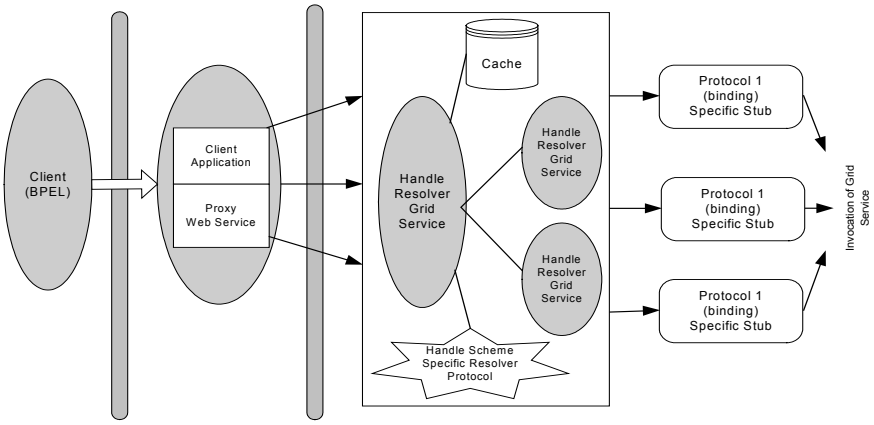

Figure 1: The Proposed Architecture

Thus, the GRS reference is stored as a global variable and is visible to the whole instance. When BPEL4WS wishes to call individual operators in the Grid service, it calls the BPEL4WS engine, such as BPWS4J or Collaxa, to activate Proxy Web Services stored in the Web service container. The Proxy Web Service then uses the Grid service reference, which is stored in the public attribute, to tell the Grid service container to invoke the corresponding Grid services. The Grid service replies with its results to the Grid service client that made the request. The Grid service client passes the response to its Proxy Web Service. The BPEL4WS engine can obtain the result and pass it on to the next service. This architecture is illustrated in Figure 1.

This principle can be used to design a Grid service from existing BPEL4WS descriptions. The advantage of this approach is that the impact on the BPEL4WS descriptions and the associated WSDL can be minimised when the Grid service is re-deployed to different locations.

In order to examine the feasibility of the proposed architecture, we use a simplified *heartbeat* example, which is based on Model-View-Control (MVC) [13] design patterns. The example is implemented such that it represents three Grid services. As shown in Figure 2, the client includes an interface for users to see the number of heartbeats and control the speed of heartbeat.

The client triggers the Heart View service to invoke other Grid services and to receive the output from the Heart Model. The Heart Control model (a Grid service) receives the request from the Heart View model and passes it to the Heart Model. A fragment of WSDL generated from the GT3 that describes the Heart Control model is shown in the Appendix. Similarly, the Heart Model is a Grid service that takes the request from the control, responds to it and sends the result to the Heart View model (see Figure 2).

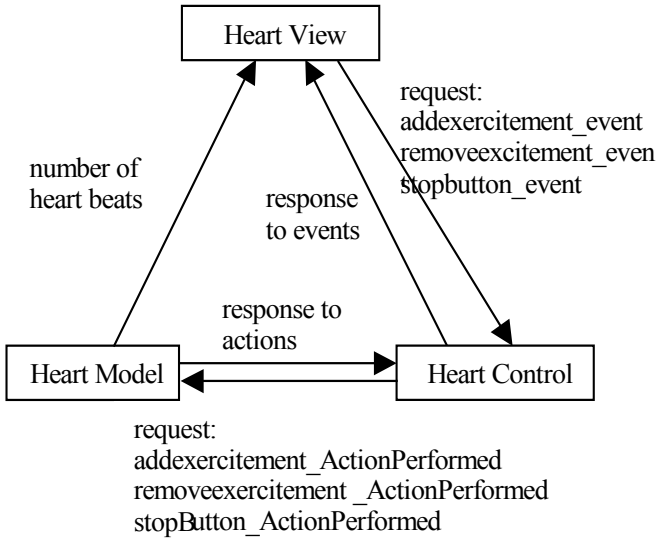

Figure 2. Composition of Grid services

Web services, corresponding to three Grid services, are specified as partners. Three possible events are defined (add, remove and stop) as an XML complex type, and are implemented as Java beans. Thus BPWS4J starts up the Web services, which in turn call the Java beans. These Java beans invoke the corresponding operators in the Grid services.

```
<process name="HeartbeatModelling"
----
  <variables>
    <variable name="request"
            messageType="tns:MVC"/>
```

```
  <variable name="HeartContol"
          messageType="tns:MVC"/>
 </variables>
 <partners>
  <partner name="contoller"
         serviceLinkType="lns:HeartControlLinkType"
         myRole="controller"/>
  <partner name="requester"
         serviceLinkType="lns:HeartViewLinkType"
         myRole="viewer"/>
  <partner name="modeller"
         serviceLinkType="lns:HeartModelLinkType"
         partnerRole="modeller"/>
 </partners>

 <Sequence>
  <receive name="initial" partner="viewer"
         portType="view:HeartViewPT"
         operation="start" variable="request"
         createInstance="yes">
  </receive>
  <invoke name="heartcontrol" partner="controller"
        portType="control:HeartControlPT"
        operation="Performedaction"
        inputVariable="request"
        outputVariable="actionPerformed">
   </invoke>
  <assign name="assign">
   <copy>
          <from variable = "actionPerformed"
           portType="control:MVC" />
          <to variable="HeartControl" PortType = "tns:MVC"/>
   </copy>
  </assign>
--------
  </Sequence>
</Process>
```

Figure 3. A Code fragment of BPWSJ

4. Analysis and discussion

Experiments carried out on the prototype system show the feasibility of the proposed architecture — using BPEL4WS for Grid service composition. The experiments also revealed a number of similarities between BPEL4WS and Grid services. That is, BPEL4WS and Grid services share a number of similar properties such as life cycle management and stateful interactions.

The experiments also revealed the following main differences between BPEL4WS and Grid services:

(1) Coordination between Web service instances is driven by the data in BPEL4WS. The way of correlating Web service instances in BPEL4WS is similar to database systems handling tables through index keys. Thus, developers have to define correlation sets from *portTypes* in WSDL and use them to correlate instances. On the other hand, OGSI uses Grid service instance references to coordinate Grid service instances. Each Grid service instance has a unique reference (similar to an object reference). Since GT3 is mainly implemented through the JAXRPC specification (a Web Service specification based on Java RMI), the management of a collection of instances is similar to handling multiple instances in Java. Therefore, GT3 cannot export its Grid service instance references to BPEL4WS, and BPEL4WS cannot hold references of the Grid service instances. Consequently, the additional functions in the GT3 such as the grouping of Grid services and life cycle management, pre-call, post-call grid services cannot be utilised by BPEL4WS directly. BPEL4WS does not support any construct that allows Web service instances to be destroyed. Instead, it provides termination of the whole process.

(2) The other main issue is the different serialisation approaches used in GT3 and BPEL4WS. Serialisation in Grid services is to serialise the Grid service instances, but BPEL4WS only serialises the variables in the process. Therefore, BPEL4WS cannot instruct Grid services to serialise the instances. Both BPEL4WS and GT3 refer to WSDL to obtain information about services, using this information to initiate requests and respond to them. However, the versions they build upon are different. GT3 adopts WSDL1.2, renaming it as GWSDL, and BPEL4WS is based on version 1.1. BPEL4WS cannot parse the extra features proposed in WSDL 1.2. This issue may easily be resolved when WSDL 1.2 becomes an official WWW specification. One of important features that GT3 supports is the notification mechanism. It is similar to the observer and observable mechanism in Java, allowing services to push or pull information when the state changes. However, no mechanism in BPEL4WS can map to this mechanism.

We make the following observations from the above analysis:

- It is not a trivial task to design a specification or system that enables BPEL4WS and OGSI infrastructures to be fully integrated. Even though they have different focuses, they should have consistent infrastructures to explore their potentials.
- BPEL4WS is not the only specification for orchestrating Web services. The Semantic Web community has proposed DAML-S [14] for describing the semantics of Web services and composition mechanisms for Web services. However, there is no sophisticated engine like BPWS4J or Collaxa to support the DAML-S specification. [15] defines the semantics of Web services via DAML-S and translates the descriptions to BPEL4WS. Thus, BPWS4J can

provide a run-time environment to execute Web services accordingly. Other on-going research is to employ agents with a specific reasoning mechanisms, such as GoLog [16], to compose the Web services.

5. Conclusions

In this paper, we proposed an architecture that enables the composition of Grid services using BPEL4WS. The proposed architecture provides a high-level bridging between BPEL4WS and OGSI, but it does not attempt to fully integrate their infrastructures due to reasons stated above. An MVC design pattern was used to test the feasibility of the proposed architecture. Experiments show that the proposed architecture is adequate for modelling or orchestrating Grid services using BPEL4WS. Based on our experiments we provided a detailed analysis of the issues related to the mis-match between OGSI and BPEL4WS. It is observed that such issues must be addressed so as to use BPEL4WS in Grid service composition. Failing to do so may impede applications that require more controllable power over Grid service instances or try to utilise the features supported by the GT3. In the future, we plan to extend our architecture in order to tackle the issues identified in this paper.

6. References

[1] D. J. Mandell, and S. A. McIlraith, "A Bottom-Up Approach to Automating Web Service Discovery Customization, and Semantic Translation", *The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web,* Budapest, 2003.

[2] F. Curba, R. Khalaf, N. Mukhi, S. Tai, & S. Weerawarana, The Next Step in Web Services, *Communication of ACM*, 46(10), 2003, pp29-34.

[3] W3C Note "Web Services Definition Language (WSDL) 1.1", http://www.w3.org/TR/WSDL

[4] W3C Note "Simple Object Access Protocol (SOAP) 1.1", http://www.w3.org/TR/WSDL

[5] UDDI. The UDDI technical white paper, http://www.uddi.org/, 2000.

[6] OGSI, Open Grid Services Infrastructure (OGSI) Version 1.0, http://www-unix.globus.org/toolkit/ documentation.html

[7] Globus toolkits 3, http://www-unix.globus.org/toolkit/documentation.html

[8] Business Process Execution Language for Web Services Version 1.1, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[9] Web *Services Architecture*, http://www.w3.org/TR/ws-arch.

[10] Service-*Oriented Architecture (SOA) Definition*: http://www.servicearchitecture.com/web-services/articles/serviceoriented_architecture_soa_definition.html.

[11] S. Parastatidis, J. Webber, P. Watson, & T. Rischbeck, " A Grid Application Framework based on Web Services Specifications and Practises", http://www.neresc.ac.uk/projects/gaf/, 2003

[12] Web Services *Description Language (WSDL) Version 1.2,* Published W3C Working Draft, World Wide Web Consortium, http://www.w3.org/TR/wsdl12/

[13] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, 1998

[14] DAML Services Coalition. DAML-S: Semantic Markup for Web Services. DAML-S v. 0.9 White Paper, http://www.daml.org/services/daml-s/0.7/daml-s-wsdl.html, Sept 2003.

[15] Evren Sirin, James Hendler, Bijan Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions." *Proceedings of "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003*, 2002.

[16] S. McIlraith and T. Son. Adapting Golog for Composition of Semantic Web Services. Conference Proceedings on Knowledge Representation and Reasoning, April 2002.

Appendix:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://hello.gt3tutorial/heartcontrol"… />
  <wsdl:message name="stopButton_ActionPerformedRequest">
    <wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message
name="removeExcitement_ActionPerformedRequest">
    <wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="addExcitement_ActionPerformedRequest">
    <wsdl:part name="in0" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="HeartControlPortType">
-----
    <wsdl:operation name="removeExcitement_ActionPerformed"
parameterOrder="in0">
      <wsdl:input
message="impl:removeExcitement_ActionPerformedRequest"
name="removeExcitement_ActionPerformedRequest"/>
---------
      <wsdl:input
message="impl:addExcitement_ActionPerformedRequest"
name="addExcitement_ActionPerformedRequest"/>
    </wsdl:operation>
  </wsdl:portType>
…..
    <plnk:partnerLinkType name="HeartControl">
      <plnk:role name="HeartController">
        <plnk:portType name="tns:HeartControlPortType"/>
      </plnk:role>
  </plnk:partnerLinkType>
</wsdl:definitions>
```