Grid Service Composition in BPEL for Scientific Applications

Onyeka Ezenwoye[1], S. Masoud Sadjadi[2], Ariel Cary[2], and Michael Robinson[2]

[1] Electrical Engineering and Computer Science Department
South Dakota State University, Brookings, SD 57007
onyeka.ezenwoye@sdstate.edu
[2] School of Computing and Information Sciences
Florida International University, 11200 SW 8th Street, Miami, FL 33199
{sadjadi, acary001, mrobi002}@cs.fiu.edu

Abstract. Grid computing aims to create an accessible virtual supercomputer by integrating distributed computers to form a parallel infrastructure for processing applications. To enable service-oriented Grid computing, the Grid computing architecture was aligned with the current Web service technologies; thereby, making it possible for Grid applications to be exposed as Web services. The WSRF set of specifications standardized the association of state information with Web services (WS-Resource) while providing interfaces for the management of state data. The Business Process Execution Language (BPEL) is the leading standard for integrating Web services and as such has a natural affinity to the integration of Grid services. In this paper, we share our experience on using BPEL to integrate, create, and manage WS-Resources that implement the factory pattern. To the best of our knowledge, this work is among the handful approaches that successfully use BPEL for orchestrating WSRF-based services and the only one that includes the discovery and management of instances.

Keywords: BPEL, Grid Computing, WSRF, OGSA-DAI, Service Composition.

1 Introduction

Grid computing promises to harness the resources available on disparate distributed computing environments to create a parallel infrastructure that allows for applications to be processed in a distributed manner. The goal is to create an accessible virtual supercomputer by integrating distributed computers with the use of open standards [1]. To this end, the Open Grid Services Architecture (OGSA), developed by Global Grid Forum (GGF), defines an architecture for service-oriented Grid computing; GGF no longer exists, but the OGSA architecture is still current. This architecture utilizes Web services standards such as XML, SOAP and WSDL [2].

Under the OGSA, computational and storage resources are exposed as an extensible set of networked services that can be aggregated to create higher-function applications. These Grid services, adhere to a set of OGSA-defined conventions for creation, lifetime management, discovery and change management [3]. Aligned with these conventions is the Web Services Resource Framework (WSRF). WSRF is a set of specifications that are defined in terms of existing Web services technologies, for modeling and management of stateful resources. The specification defines a set of interfaces that Grid services may implement. These interfaces which address issues like dynamic service creation, lifetime management, notification, and manageability, allow applications to interact with Grid services in standard and interoperable ways [4].

Key to the realization of the benefits of Grid computing is the ability to integrate basic services to create higher-level applications. We argue these higher-level applications will provide the right level of abstraction for the non-computer scientists. Thus, allowing them to concentrate on their domain specific work instead of the technical issues of integrating tools. Workflow languages permit such aggregation of services. With such languages, higher-level application can be modeled as graphs where the nodes represent tasks while the edges represent inter-task dependencies, data flow or flow control. Tasks may be performed by basic services. The Business Process Execution Language (BPEL) [5] has become the leading language for the aggregation of Web services. In this paper, we share our experience in using BPEL to compose WSRF-based Grid services to create a bioinformatics application for protein sequence matching. We show how BPEL can be used to interact with WSRF-based services that implement the factory/instance pattern.

The rest of this paper is structured as follows. Section 2 covers WSRF and some of its component specifications. Section 3 presents the bioinformatics application and Section 4 shows how BPEL is used to integrate Grid services to create the application. Sections 5 provide some conclusion.

2 Web Services Resource Framework

In 2003, the Global Grid Forum, a working group for the standardization of Grid computing, released the specification for the Open Grid Services Infrastructure (OGSI). OGSI is a set of conventions and extensions on the use of Web Service Definition Language (WSDL) and XML Schema to enable the modeling and management of stateful Web services. The OGSI specification addresses issues concerning creation and management of the lifetime of instances of services, declaration and inspection of service state data, notification of service state change and standardization of service invocation faults.

In 2004, OGSI was refactored into the Web Services Resource Framework (WSRF). This framework standardizes the concept of Web Services Resource (WS-Resource) [6], which is, the association of a state component with a Web service. This association permits, through *standardized interfaces*, the manipulation of the *named typed* state component as part of the execution of the Web service. This creates the impression of statefulness of that Web service.

WSRF addresses some of the criticisms [7] of OGSI such as; the specification was too monolithic and did not allow for flexible incremental adoption and

Specification	Description
WS-BaseFaults	Defines a set of fault types
WS-RenewableReferences	Defines means for renewal of invalid references
WS-ResourceProperties	Defines the representation of the properties of a stateful resource
WS-ResourceLifeCycle	Defines means for resource creation and destruction
WS-Notification	Defines mechanisms for event subscription and notification
WS-ServiceGroup	Defines primitives for managing collections of services

Fig. 1. WSRF component specifications

extensions to WSDL 1.1. OGSI was also seen as too object-oriented by coupling the service and the stateful resource it acts upon as one entity. WSRF maintains all the functions of OGSI but incorporates some existing Web services technologies. WSRF partitions its functionality into distinct component specifications (as shown in Figure 1). With this separation, developers can now choose which of the specifications to use. WSRF now supercedes the initial OGSI specification, thereby rendering it obsolete.

An instance of a stateful resource may be created by the use of a WS-Resource factory. This factory is any Web service capable of instantiating the stateful component. To instantiate a stateful resource, the Web service has to create a new stateful resource, assign an identity to that resource and create the association between the resource and its Web service. The factory returns an *endpoint reference*, which contains the identifier that refers to the new stateful resource.

3 WSRF Services for Bioinformatics

In this sections, we use a Grid application that we developed for computational biology as the case study to demonstrate the use of BPEL in the orchestration of WSRF-based Grid services. This application attempts to match protein sequences [8]. This matching of sequences can be computationally intensive

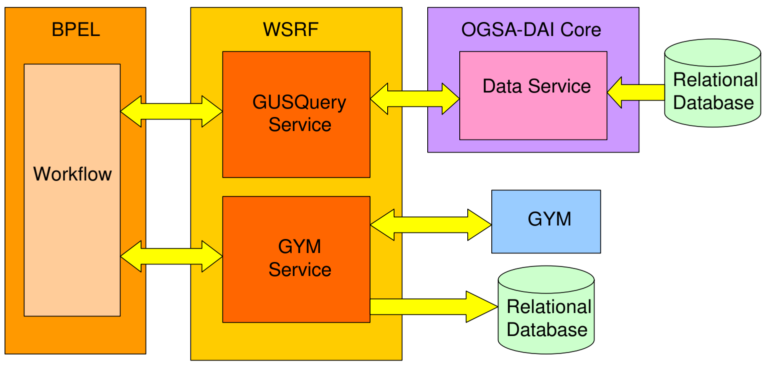

Fig. 2. The high-level architecture of the application

depending on the size of the sequences processed. Figure 2 shows the high-level architecture of the application. Below, we explain some of the components of this architecture.

OGSA-DAI. There is a need to seamlessly access disparate sources of biological data and integrate them into the Grid for further processing. OGSA-DAI [9] is middleware that facilitates the access and integration of data from separate sources in a Grid computing environment. OGSA-DAI makes data sources accessible via Web services (Data services).

GYM. GYM is a biological application for processing protein data sequences. Here, GYM is used to detect Helix-Turn-Helix (HTH) Motifs [8] in protein sequences. The GYM program is a legacy application written in C. It takes as input, a sequence of protein data. GYM instances are run on Grid nodes to process the sequences available from the sources.

GUSQuery Service. This is a WS-Resource. The resource in this case is the protein sequence data obtained from an OGSA-DAI data service. This service contains a search method which takes as input a range of sequences and the location of the data service through which to get those sequences. This service is accompanied by a *factory* service called *GUSQueryFactory*. The factory contains a create method which creates an instance of the GUSQuery Service.

GYM Service. This is also a WS-Resource. It contains a method which takes as input a range of protein sequence data. This data is then processed locally using a GYM application. The result from the GYM application is stored in a database. This service also has a factory service called *GymFactory*.

Workflow. This BPEL process weaves together the interaction between the GUSQuery service and the GYM service. This executable workflow, which is exposed as a Web service, is also responsible for creating the instances of those services through their respective factory services. It uses the endpoint references returned by those factories to identify the specific service instances. It passes the protein sequence data from the GUSQuery service to the GYM service. The first step is to use the GUSQueryFactory to create an instance of GUSQueryService, this operation returns the endpoint reference identifier for the instance. The GUSQueryService instance is then invoked to retrieve a set of protein sequences. These sequences are then sent to the GymService for processing , just after an instance of that service is create. The last step retrieves the result of that sequence processing.

4 WSRF with BPEL

In this section, we show how the interaction with the WSRF-based services is achieved in BPEL. The details about the definition of the services themselves, is outside the scope of this paper. Due to page limitations, some code have been simplified or detail eliminated. For a more detailed version of the content in this paper, please refer to our technical report [10].

4.1 Creating a Web Service Instance

Creating a new Web service resource instance involves making a call to the *createResource* operation of designated factory service. This is achieved by using BPEL's service invocation mechanism. The `<invoke>` construct allows a BPEL process to invoke a one-way or request-response operation on a portType (interface) offered by a partner service [11]. Using this construct, an invocation to the `createResource` operation of the GUSQuery factory service is made.

The invoke activity which makes a synchronous call to the factory service, contains the `portType` of the operation as well as the `inputVariable` and `outputVariable` variables. If the invocation is successful, the `outputVariable` will contain the endpoint reference of the created instance.

4.2 Invoking the Web Service Instance

Since the identifier of a WS-Resource instance is obtained at runtime, any message to this instance must contain the resource identifier in its SOAP header. The BPEL specification allows for the actual service endpoint of a partner to be dynamically defined within the process. The specification however, does not make provisions for how dynamically obtained information such as resource identifiers can be define for those endpoints. This type of information needs to be mapped to the headers of the SOAP messages for the target endpoint. Because the BPEL specification is deficient in this regard, the method mapping desired information to SOAP headers depends on the specific implementation of the BPEL execution engine. The method me describe below is suited for the ActiveBPEL Engine.

To dynamically associate an endpoint reference to a service, the WS-Addressing endpoint reference [12] is used to represent the dynamic data required to describe a partner service endpoint [11]. To achieve the association of a partner with its service endpoint, an endpoint reference has to be assigned to the declared partner link within the process. As shown below, we use the `copy` operation of an assignment activity to copy *literally* an endpoint reference to a variable (`DynamicEndpointRef`).

```
<copy>
   <from>
      <wsa:EndpointReference xmlns:s="...">
         <wsa:Address/>
         <wsa:ServiceName PortName="GUSQueryPortType">
            s:GUSQueryService
         </wsa:ServiceName>
         <wsa:ReferenceProperties>
            <!--Elements to be mapped to the SOAP Header-->
            <wsa:Action/>
            <wsa:To/>
            <wsa:From/>
            <ns2:GUSQueryResourceKey/>
         </wsa:ReferenceProperties>
      </wsa:EndpointReference>
   </from>
   <to variable="DynamicEndpointRef"/>
</copy>
```

This endpoint reference contains an `Address` element that will hold the service endpoint address. The `ReferenceProperties` of the endpoint reference contains some WS-Addressing message information header elements and a `GUSQuery-ResourceKey` element. The `GUSQueryResourceKey` element will hold the resource identifier for the WS-Resource. Values for the endpoint reference will be assigned at run time. The message information header elements and the `GUSQuery-ResourceKey` will be mapped, by the BPEL engine to the invocation SOAP message for the partner Web service, which in this case is `GUSQueryService`.

The WS-Resource identifier information required for the endpoint reference is copied from the reply message of their respective factory services. The copy operation below copies the service endpoint address from the factory response message (`CreateResourceResponse`) to the endpoint variable (`DynamicEndpointRef`). The `query` attribute of the `<from>` and `<to>` clauses are XPath [13] queries. XPath queries are used to select a field within a source or target variable part.

```
<copy>
   <from variable="CreateResourceResponse"
      part="response"
      query="/ns4:createResourceResponse
               /wsa:EndpointReference/wsa:Address"/>
   <to variable="DynamicEndpointRef"
      query="/wsa:EndpointReference
               /wsa:ReferenceProperties/wsa:To"/>
</copy>
```

A similar mechanism is used to assign the service endpoint address to the `<wsa:Address>` property of the endpoint reference variable. The BPEL engine needs this address to determine the destination of the invocation message for the service. The `<wsa:To>` component of the message information header is used by the service to determine the endpoint of the required service instance. We use the same address returned by the factory because for this application, the address of a service and its instance are the same.

The name of the operation to be invoked on the WS-Resource instance needs to be assigned to the `Action` part of the SOAP header. To achieve this, an XPath expression to write the name as a string to the endpoint reference variable. An XPath expression, which is specified in an expression attribute in the `<from>` clause, is used to indicate a value to be stored in a variable. The string that represents the operation, is in the for of a URI that includes the target namespace of the WSDL document for the WS-Resource and the associated portType. Thus in the listing below, `http://GUSQueryService_instance` is the namespace, `GUSQueryPortType` is the portType and `searchSequence` is the operation.

```
<copy>
   <from expression="string('
      http://GUSQueryService_instance
      /GUSQueryPortType/searchSequence')"/>
   <to variable="DynamicEndpointRef"
      query="/wsa:EndpointReference
      /wsa:ReferenceProperties/wsa:Action" />
</copy>
```

The listing below shows how we use the `copy` operation and XPath queries to copy the resource instance key (`GUSQueryResourceKey`) from the factory response message to the endpoint reference variable.

```
<copy>
   <from variable="CreateResourceResponse" part="response"
      query="/ns4:createResourceResponse
      /wsa:EndpointReference/wsa:ReferenceProperties
      /ns2:GUSQueryResourceKey"/>
   <to variable="DynamicEndpointRef"
      query="/wsa:EndpointReference/wsa:ReferenceProperties
      /ns2:GUSQueryResourceKey"/>
</copy>
```

The `<wsa:From>` property of the message information header identifies the source of the meassage, this property can be set with the WS-Addressing "anonymous" endpoint URI [12].

After assigning values to all the necessary parts of the endpoint reference variable, an association is now made with this variable and the desired partner link. As shown below, a `copy` operation is used to copy the endpoint reference variable (`DynamicEndpointRef`) to the predefined partner link. An invocation can now be made to the Web service (WS-Resource) partner (GUSQuery service). The information carried in the SOAP message header of the invocation is used to identify the appropriate instance of this service.

```
<copy>
   <from variable="DynamicEndpointRef"/>
   <to partnerLink="gus"/>
</copy>
```

4.3 Accessing Resource Properties

The *WS-ResourceProperties* specification includes a set of port types for querying and modifying the state of a WS-Resource. The Gym service (Section 3) implements the *GetResourceProperty* port type of this specification. We use this port type and its operation (also called *GetResourceProperty*) to access the result from the Gym application (Section 3). Prior to invoking the *GetResourceProperty* operation, some initialization needs to be made to the variable of its input message. This initialization includes the name of the resource property to which we want to retrieve the value. In our case, this resource property is called *result.* The listing below shows how we initialize the *GetResourcePropertyRequest* in the BPEL process. The `<from>` clause includes (as attributes) the target namespace of the WSDL documents that contain the definitions for the *GetResourceProperty* port type and the *result* resource property.

```
<copy>
   <from>
      <GetResourceProperty
      xmlns="http://docs.oasis-open.org/wsrf/2004/06
      /wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
      xmlns:ns3="http://GymService_instance">
         ns3:result
      </GetResourceProperty>
   </from>
<to variable="GetResourcePropertyRequest"
      part="GetResourcePropertyRequest"/>
</copy>
```

Because we are trying to access the resource properties of a WS-Resource instance, assignments need to be made to all parts of the message header necessary for identifying the instance. The way to do this is described in Section 4.2, the only difference now is in the URI that specifies the verb of the invocation message.

5 Conclusion

In this paper, we discussed and explained how BPEL can be used as a language for integrating WSRF-based Grid services. In a case study, we demonstrated how some WSRF-based Grid services can be integrated to create a Bioinformatics application. The integrated WSRF services implement the factory pattern. We showed how BPEL can be used to create, discover and manage WS-Resource instances. The centralized nature of data movement in BPEL presents a problem for high-performance computing, however, this limitation can be remedied by using techniques that enable the direct transfer of data between partner services. Also, the BPEL specification does not make provisions for how dynamically obtained information such as resource identifiers, usernames and passwords can be specified within SOAP message headers. The method of achieving this is left open to the implementation of the various BPEL engines. Therefore, there is a need for standardization in this regards for BPEL process to remain portable and assume its place as the language for orchestrating Grid services.

Further Information. A number of related papers and technical reports of the Autonomic Computing Reserach Laboratory can be found at the following URL: `http://www.cs.fiu.edu/~sadjadi/Publications/`.

Acknowledgements. This work was supported in part by IBM, the National Science Foundation (grants OCI-0636031, REU-0552555, and HRD-0317692).

References

1. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the Grid: Enabling scalable virtual organizations. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, Springer, Heidelberg (2001)
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. W3C. 1.1 edn. (2001)
3. Foster, I., Kesselman, C., Nick, J.M., Tuecke, S.: Grid services for distributed system integration. Computer 35(6), 37–46 (2002)
4. Web Services Resource Framework, `http://www.globus.org/wsrf/`
5. Ezenwoye, O., Sadjadi, S.M.: Composing aggregate web services in BPEL. In: Proceedings of The 44th ACM Southeast Conference, Melbourne, Florida (2006)
6. Foster, I., et al.: Modeling stateful resources with Web services (2004)
7. Czajkowski, K., et al.: From OGSI to WS-Resource framework: Refactoring and evolution (2004)
8. Narasimhan, G., et al.: Mining for motifs in protein sequences. Journal of Computational Biology 9(5), 707–720 (2002)

9. The OGSA-DAI Project, http://www.ogsadai.org.uk/
10. Ezenwoye, O., Sadjadi, M., Carey, A., Robinson, M.: Orchestrating wsrf-based grid services. Technical report, School of Computing and Information Sciences, Florida International University (2007)
11. Andrews, T., et al.: Business process execution language for web services version 1.1 (2003)
12. Web Services Addressing (WS-Addressing), http://www.w3.org/Submission/ws-addressing/
13. XML Path Language (XPath), http://www.w3.org/TR/xpath