End User Oriented Ad Hoc Web Services Composition in a Scientific Application Portal Architecture

Xiaorong Xiang Gregory Madey
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
xxiang1@nd.edu, gmadey@nd.edu

Abstract

Web services technologies provide e-Business a way to automate and streamline distributed business processes. They are also emerging as a new approach for supporting e-Science by providing access to heterogeneous computation resources and integration of distributed scientific applications. In this paper, we present a scientific application portal architecture that enables end users to compose a new scientific task by integrating a set of high level web services provided by portal developers. It is also intended to provide end users high quality computation services by choosing an optimal computing resource. The portal design demonstrates that the flexibility, reusability and interoperability can be provided for both end users and portal developers by using web services technologies.

1. Introduction

Web services technologies are emerging as an approach to integrate business applications over the Internet and Intranet. Web services composition is the process of combining a set of simple web services to accomplish a larger and sophisticated task or business process. The most popular business application used for demonstration these web services composition technologies is "Traveler Planner". A travel planner aggregates multiple component services for flight booking, travel insurance, hotel booking, car rental, and itinerary planning, which are executed sequentially or concurrently. Selection and binding of component services can be carried out at application design time, or during the execution of a composition services according to multiple criteria to approach optimality [18].

The description of interactions, exchanging messages, and execution sequence among web services are a crucial problem for seamless process integration across enterprise boundaries. WSDL, which specifies the syntax of the input and output messages of a software component, is not sufficient to describe the interactions between Web services. Therefore, industries propose various web services flow specification standards, such as BPFL4WS, BPEL, WSFL, BPML, WSCI, and XLANG, driven by "concrete products and/or commercial interest" [12]. Users have to use specific vendor pattern for their services composition. Another research stream on web services description, called semantic web services [8], is emerging in the research community. It is focused on extending the vision of the semantic web [3] to describe, discover, and compose web services. A set of languages, such as RDF, DAML+OIL, DAML-S, and OWL, have been developed to represent the properties of web services using terms from the domain specified ontology. At the current stage, the semantic approach is mainly conducted in academic research with limited implementation and product support.

Web services technologies provide e-Business a way to automate and streamline distributed business processes. It is also emerging as a new approach for supporting e-Science by providing access to heterogeneous computation resources and integration of distributed scientific applications. One of the active research areas is the integration of web services technologies with grid computing technologies, such as the Globus toolkit [11], to providing high performance computational environments [9] [10].

Most web services composition research focuses on autonomous aggregation of services into a backend process by using or extending the technologies discussed above. The design and deployment process of workflows typically has to be done by IT experts. A web based scientific application portal is intended to provide scientists a unified place to use and share distributed information resources, software components, and computation resources. End users of a scientific application portal are scientists who are experts in their research domain but often novices in computer science. Unlike end users of business applications, scientists need capabilities to control the modeling and execution process of their scientific tasks in order to meet their own research goals.

In this paper, we present a web based scientific application portal architecture building upon existing web services standards and protocols. A set of web services is developed along high level user interfaces for supporting the scientific modeling and data sharing in a scientific domain. End users can share the data information that is stored in remote databases by accessing these web services individually. The portal allows end users to specify a workflow for describing a particular scientific task by selecting service components without dealing with the detail description of web services. The submitted workflow is executed on a workflow engine provided in the portal. Both synchronous and asynchronous service invocations are supported for short running and long running jobs. Also the portal is intended to provide quality services for end users by choosing optimal computing resources for long running jobs. End users can also manage the workflow executing process through a web browser. The portal design demonstrates that flexibility, reusability and interoperability can be provided for both end users and portal developers by using web services technologies.

The rest of the paper is organized as follows. Section 2 summarizes some related works on web services technologies and a motivating scientific application. Section 3 outlines the web service based portal architecture. Section 4 presents elementary web services created in the portal. Section 5 describes the development and specification of end user visible services. Section 6 presents a web service composition framework that can be used for specifying and executing workflow. Section 7 draws conclusions and presents future plans.

2. Related work

2.1. Motivating scientific application

An example scientific domain that we use to illustrate the design of the web services based portal architecture is studies of Natural Organic Matter (NOM). NOM is a mixture of molecular compounds with different types of structures, compositions, functional group concentrations, molecular weights, and different degree of reactivity. NOM comes from animal and plant material in the natural environment. It exists everywhere in the world, from terrestrial ecosystems to aquatic environments. NOM plays a crucial role in the evolution of soils, the transport of pollutants, and the global biochemical and geochemical cycling of elements. The evolution of NOM over time from precursor molecules to mineralization is an important research area in a wide range of disciplines, including biology, geochemistry, ecology, soil science, and water resources. NOM, a prevalent constituent of natural waters, is highly reactive with mineral surfaces. While NOM is transported through soil pores by water, it can be adsorbed onto or desorbed from mineral surfaces. Sorption of NOM is an important consideration in the treatment of drinking water. The details of the NOM project are described in [14][2].

In order to provide scientists a robust platform for modeling, simulating, and better understanding the evolution behavior of NOM over time, a web based scientific application portal needs to be provided. The portal architecture should be able to let scientists formulate their application easily and run them in a heterogeneous distributed computational environment without considering the details of hardware and software configuration that are not relevant to their applications. This portal should also be able to let scientists from multiple disciplines to share their data and information, analyze the application results, and retrieve the data from the data repositories.

2.2. Web services architecture

Compared to earlier distributed computational technologies such as DCOM, CORBA, and EJB, web services overcome the limitations they have, such as platform dependency, tight coupling and limited interoperability. WSDL, UDDI and SOAP are three basic standards for the web services technologies. Web services, platform independent software components, are created by service providers and expressed by the specification of a Web Services Description Language (WSDL). Service providers may store the description of a particular service into a public repository (e.g. UDDI). Service requestors can search a particular web service in the repository or access the service directly if they know the endpoint of the web service.

Web services are a feasible paradigm for building scientific application portals. A web services based portal architecture can reduce software development costs, improve software quality, and use distributed computational resources efficiently. It enables portal developers to provide new services to users more easily. There exist many scientific applications written in Fortran, C and C++ running on Windows or Unix machines. With web services technologies, portal developers can integrate these applications with few changes. It also assists portal developers integrating web services provided by other service providers [15] into a portal with end user preferred interfaces.

2.3. Web services composition

Simple interactions of web service providers and requestors through standard message and protocols are not enough to integrate web services on the Internet or Intranet. A set of workflow specification languages is emerging to support the data flow and control flow representation. Among all of these specifications, BPEL4WS is the most mature and widely supported by the industry and research community. Service compositions described in the BPEL4WS format may be deployed on execution engines, such as BPWS4J [4] and Collaxa BPEL server [6]. Composing a business process with BPEL4WS normally needs a web services composition tool. The deployment and running of the workflow are normally dependent on specific web servers; thus, end users of scientific application portals cannot be expected to handle the development process.

Along with these industry supported workflow specifications, various frameworks, prototypes, tools are proposed to deal with the automatic web services search and composition. [17] presents a Petri Net based process modeling technique for composition of web services. [5] extended the WebML to describe complex processes. [16] proposed a packaging mechanism for composing existing Web services to support the reuse, specialization, and extention of defined service components. [1] present a system that allows clients to control and monitor job execution in the Grid community. Most of the research is work in progress and suitable for use only in specific research domains. It is hard to extend them into our portal design without significant changes or extra implementations. The round trip interactions between end users and the process cannot be handled easily while the composed process is executing. We present a simple, extensible, and end user oriented workflow specification in the portal architecture to address these issues.

3. Overview of the Scientific Application Portal Architecture

The purpose of a web based scientific application portal is to provide scientists an approach to access software and data information that reside on a distributed computational environment via a standard web browser. With the portal architecture, users can use the high performance computing instruments that are not affordable by a small research group. Users can also access software that they are authorized to use without the extra work of downloading and installing on their local machines.

Main functions that a portal should provide to end users are:

- Secure access: Users should be get authentication when they sign up and be assigned authorization to the resource and services that they are permited to use.
- Application formulation and management: The portal should be able to help users formulate their applications by providing a configuration parameters editor, execute their applications, access the status of running applications, and stop the applications if it is necessary.
- Data analysis tools: The portal should include data analysis tools that are used to analyze the application results.
- Information services: Users should be able to request data information from the data repository or other information resources. They should also be able to send data information to the repository and use them at a later point in time. The information can be displayed as HTML pages or represented as downloadable files that users can analyze using other tools. Users can also upload the results to the repository and to share with other users.

The portal architecture prototype is built upon the existing web services standards and technologies. Portal developers may implement these services that can realize functions described above from scratch or integrate services provided by other service providers. Figure 1 presents basic properties of the web services based scientific portal architecture. End users, scientists, sign up to the portal and access authorized services from user interfaces. User interfaces that are normally JavaServer Pages, send requests to corresponding service clients. Service clients are service requestors that assemble input parameters for web services using input from users and send SOAP requests to web services. Web services invocation can be accomplished in synchronous and asynchronous ways. Service clients wait for operations to be completed in synchronous invocation approach, while clients can invoke services and receive results later in the asynchronous invocation approach. The user interfaces and service clients that are responsible for requesting one or multiple web services are built as a web application. The web application can reside on the same web server or different web server as web services. The web services provided in this portal may be written in multiple programming languages and running on various platforms.

In the service oriented architecture, a large scientific process can be exposed as a web service. Also components that compose the scientific process can be separated as several web services. In order to provide a full functional portal without redundant coding, we expose pieces of scientific logic as a set of elementary web services. Elementary web services defined in this paper are autonomous units that can be processed separately to accomplish a simple task. An elementary web service is described as a WSDL specification and stored in a service repository (e.g. UDDI). It can also be combined with other elementary services to accomplish a more complex task by portal developer. End users of the scientific application portal do not need to have knowledge of SOAP or WSDL specifications. A service client, implemented by portal developers, handles interpretation of

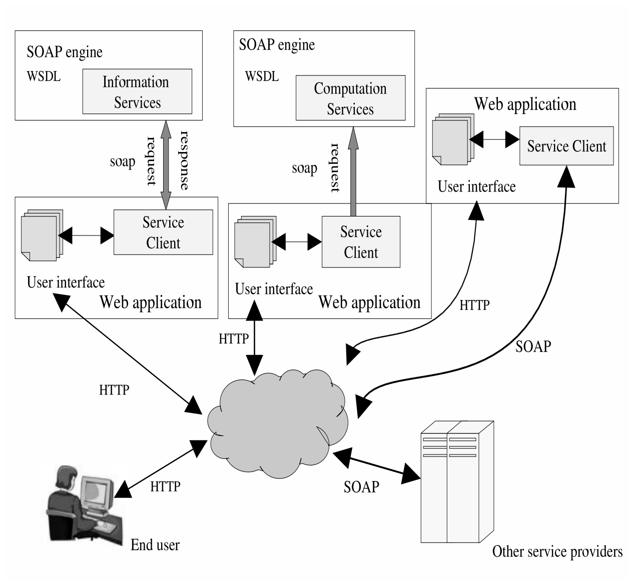

Figure 1. Web services based scientific portal architecture

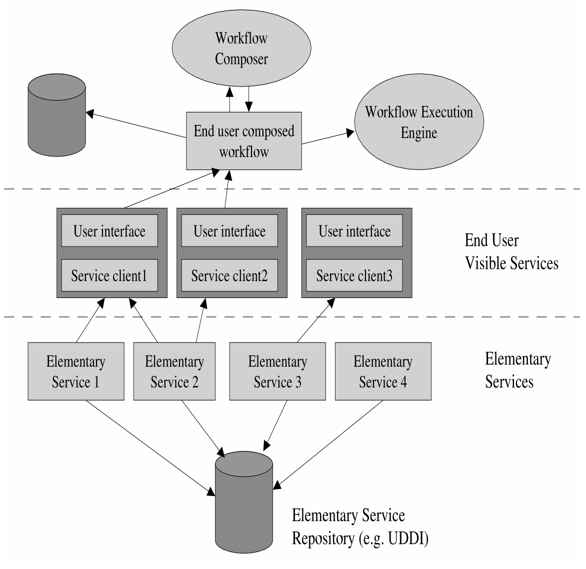

Figure 2. End user oriented web services composition heriarchy

WSDL that describes a particular web service, and generates and sends SOAP message to the web service. End user visible web services hides the underlying details from end users by providing service clients and user interfaces.

A scientific logic, provided by scientists who are collaborating with portal developers, may not satisfy all the end users. Therefore, the portal architecture provides another important feature by offering end users capabilities to form their own scientific logics to meet their particular research interests. End users can compose a new scientific process including data input processes, aggregation of computational components, and data output processes through a HTML form based editor or a visual tool. A services workflow composer is provided for checking the correctness of the scientific logic according to the dependencies among web services. If conflicts exist, the composer sends a message to the end users asking for a redesign. Otherwise, a XML based workflow specification is generated and stored in a database for future usage. A workflow execution engine is responsible for parsing the workflow and invokes appropriate web services. Figure 2 shows the relationship of these functions. The following sections describe each part in detail.

Among various web services development toolkits, such as Axis from Apache, Java WSDP from Sun, and WSTK from IBM, we chose Oracle Jdeveloper 9.0.4 as the main web service development tool and deploy web services mainly on the Oracle9iAS web server running on Linux. Windows .NET platform can be used to host web services using other languages.

4. Elementary web services creation

Elementary web services can be created and deployed on various platforms, such as J2EE and .NET. Elementary web services provided in the portal can be separated into several categories as shown in Figure 3. Some web services provide end user usable functions, while other web services provide the functions for monitoring and managing the computing resources. A short running service, such as data input, is deployed on one machine. A long running service that is a mostly CPU and memory bound process, such as a computation service, is deployed on multiple machines.

4.1. Information input services

XML format provides a standard way for exchanging data information between applications. A set of predefined XML schema is provided for supporting the data input into backend databases. The input data can either come from HTML forms or a XML file. Information input services accept XML documents as their input parameters and provide multiple operations to parse the XML document, create database connections, and write data into the database. If exceptions occur during the input process, the *getErrorMessage* operation is provided to return exception messages. The *getValue* operation is provided to return corresponding data.

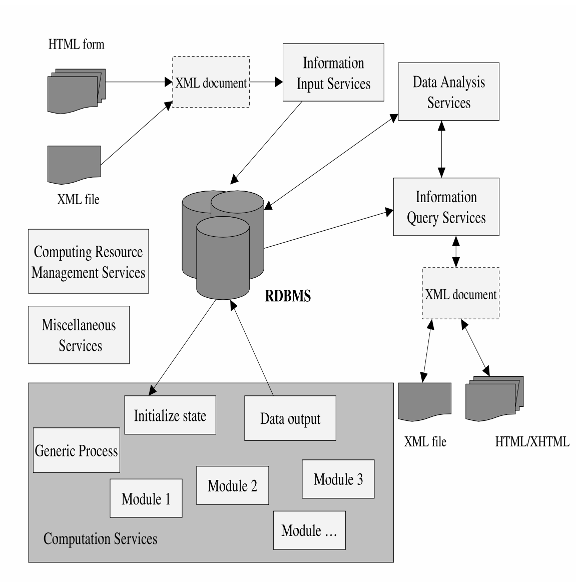

Figure 3. Elementary web services

4.2. Information query and data analysis services

Information query services accept XML documents as input parameters and provide multiple operations to create database connections, retrieve data information from databases, and generate XML documents. If exceptions occur during the input process, the *getErrorMessage* operation is provided to return exception messages. The *getData* operation is provided to return corresponding data.

Data analysis services can be incorporated with information query services to provide more useful data to end users.

4.3. Computation services

In the NOM application, a set of simulation models is provided to meet different research requirements. For each simulation model, there are several modules to simulate behavior of NOM, such as motion, sorption, and chemical reactions. It is possible to separate these modules out as web services. However, the internal computation process involves frequent iterations and internal state maintenances that would result in large data transaction inefficiency. A better strategy is to build a generic simulation model that consists of all the modules. The generic model is exposed as a computation service that consists of multiple operations that are modules from different simulation models. The computation service accepts a XML document as its input parameter. The XML document conforms to the predefined XML schema that describes modules need to be modeled and a session number responsible for identifying a task. A computation process is formed according to certain computation logic.

The scientific application portal can host various computation services. There are many existing scientific applications written in other programming languages. With a few changes, these programs can be transformed into web services using existing toolkits, such as gSOAP toolkit [7] and WSAP from systinet [13]. These programs can also be created as web services and deployed on the Windows .NET platform in our portal architecture.

4.4. Computing resource management services

In order to provide end users high quality services and efficient use of computing resources, the portal architecture provides a set of services that is used for monitoring the resource usage and managing the resource. *checkLoad* service that resides on each server returns the CPU load average and the memory usage information. This information can be retrieved using **uptime** and **procinfo** commands on Linux. CPU load average and memory usage are two main criteria that are used to choose an optimal computing resource. These services are invoked when it is time to invoke long running computation services. They are not usable for typical end users.

4.5. Miscellaneous services

A set of simple services are provided to help end users with their scientific researches. *molecule weight calculation* accepts a XML document that describes the properties of particular molecules. It calculates the molecular weight for end users. *lognormal distribution generation* accepts a XML document that describes the mean and standard deviation of molecular weight distributions for the simulation. It generates a log-normal distribution graph for users. *send email* accepts a XML document that describes the sender, receivers, subject, and email contents. It generates a message and sends to receivers.

5. End User Visible Services

The purpose of this portal architecture is to let end users, scientists, use web services or model a new simulation without knowledge of how to interprete a WSDL specification. In addition to the elementary web services, service clients and user interfaces are provided for accessing services that can be used by end users. These are called end user visible services. End users send requests to service clients by providing data from user interfaces. Service clients generate input parameters and call the remote services. Service clients also receive the output parameters from services, generate XML documents, and send back to user interfaces which render the information to end users as an HTML page.

For example, when an end user enters the portal, a set of services that the user is authorized to use is displayed. By clicking the link *create new molecule from wizard*, an interface for inputting the properties of the molecule is displayed. After the user finishes the input, by clicking the *submit* button, a request is send to a service client. The service client generates a XML document that conforms to the predefined schema describing the molecule structure and sends a SOAP request to the service. The service writes data into database and returns a XML document to the service client. The service client renders HTML pages to the end user.

The relationship between service clients and service can be one-one or one-many. A one-one relationship means that each service client maps to one service. A one-many relationship means that a service client can call several operations from multiple WSDL descriptions in a certain sequence.

End user visible services can be not only used directly, but also can be used for composing a new scientific task that end users are interested in. It is necessary to provide a mechanism that can ensure the correctness of the composed workflow provided by end users. Therefore, we first provide a XML specification that expresses dependencies among these individual web services. Four types of relationships between two individual services are specified in the portal.

- **Exclusive**: Either "Service 1" or "Service 2" can be in the workflow, but not both.
- **FullDependentOn**: If "Service 1" is in the workflow, "Service 2" must be in the workflow. Also the execution sequence must be "Service 2" then "Service 1".
- **PartialDependentOn**: If "Service 1" is in the workflow, "Service 2" is not necessary in the workflow. If "Service 2" is in the workflow, then the execution sequence must be "Service 2" then "Service 1".
- **Coexist**: If "Service 1" is in the workflow, one of the service in a group must be in the workflow. But they can execute in any order.

The relationship definition for all the end user visible services is stored in a XML file. Portal administrators can add a new service or delete a service by modify the XML document. Figure 4 shows an example of service definitions.

6. Web Services Composition Framework

When end users enter the portal, they can choose to compose a new task by clicking a button. A task editor is presented to end users. Initially, for simplicity reasons, the task editor can be a HTML form with radio buttons, text boxs,

```
<EndUserServices>
    <Service name="MoleculeDistribution">
        <UserInterface>http://host1/molculedistribution.jsp </UserInterface>
        <Category>Input</Category>
        <Request>MoleculeDistribution</Request>
        <FullDependentOn></FullDependentOn>
        <PartialDependentOn>NewMolecule</PartialDependentOn>
        <PartialDependentOn>UploadMolecule</PartialDependentOn>
        <Exclusive>AutoDistribution</Exclusive>
        <Coexist>
            <Name>NewEnvironment</Name>
            <Name>ChooseEnvironment</Name>
            <Name>UploadEnvironment</Name>
        <Coexist>
    </Service>
    <Service name="NewMolecule">
        <UserInterface>http://host2/newmolecule.jsp</UserInterface>
        <Category>Input</Category>
        <Request>NewMolecule</Request>
    </Service>
    ...
</EndUserServices>
```

Figure 4. An example of end user visable service definition

and check boxs. More sophisticated graphic composition tools can be built and provide end users a more flexible way to represent their tasks as a directed graph.

In order to illustrate the main ideas of end user oriented service composition in the portal architecture, a simple composition scenario is presented in 6.1. The service composition process and specification is discussed in 6.2. The workflow execution engine is described in 6.3.

6.1. Composition scenarios

An end user may be only interested in modeling the sorption behavior of NOM with water flow over time. The user has the mean and standard deviation of molecular weight distribution on hand, and also would like to get results to show the adsorbed molecule weight distribution over time.

The user can define the environment parameters by choosing *NewEnvironment* for creating environment parameters from a web based wizard, *UploadEnvironment* for creating environment parameters by uploading a XML file, or *ChooseEnvironment* for choosing environment parameters from an existing set up. Then the user may choose *AutoDistribution* for inputting mean and standard deviation. Two operations in the computation service, *ModelSorption* and *ModelMove*, should be choosen to be included into the computation process. *AdsorptionDistribution* and *DesorptionDistribution* should be choosen as output.

After the end user clicks the "submit" button. The data information is sent to the workflow composer.

6.2. Workflow composer

The workflow composer checks dependencies among these chosen services and operations against to the specification defined in Figure 4 using XPath. If there are conflicts, a message is sent back to the end user for correction.

After the composed task is validated, the workflow composer sends a query to the database and gets a session number as an identifier for the task. A XML document is generated to present the workflow of this task. Since the invocation for a particular service and interpretation of WSDL file is embedded in the end user visible services, the workflow specification for end user composed task only needs to specify a sequence of user interfaces without handling the details. Figure 5 shows the simple description language used in the portal for describing the workflow. The XML document is stored into a database corresponding to the task identifier. Next, it is sent to the workflow execution engine.

```
<newApplication name="xxiang1135">
  <creator>xxiang1</creator>
  <taskID>135</taskID>
  <description>Modeling the sorption properties of NOM</description>
  <sequential>
       <service name="NewEnvironment">
         <UserInterface>http://host1/NewEnvironment.jsp</UserInterface>
         <Request>NewEnvironment</Request>
       </service>
       <service name="NewMolecule">
         <UserInterface>http://host2/NewMolecule.jsp</UserInterface>
         <Request>NewMolecule</Request>
       </service>
       <service name="MoleculeDistribution">
         <UserInterface>http://host2/MoleculeDistribution.jsp</UserInterface>
         <Request>NewMolecule</Request>
       </service>
  </sequential>
  <computation>
       <service name="NOMSimu1">
         <moduleName>move</moduleName>
         <moduleName>sorption</moduleName>
       </service>
  </computation>
  <parallel>
       <service>sorptionOutput</service>
       <service>reactionOutput</service>
  </parallel>
</newApplication>
```

Figure 5. An workflow example

6.3. Workflow Execution Engine

The workflow execution engine is responsible for managing the control flow and data flow among multiple web services during the execution of the workflow. In a scientific application, a workflow submitted by a client can either be processed in synchronous or asynchronous, parallel or sequential ways.

In this portal architecture, we provide a workflow execution engine that consists of a central controller and a background processor as shown in Figure 6. The central controller is responsible to handle the synchronous web service invocation for short running jobs such as data input that requires maintain the status of interactions between end users and services. The background processor is responsible to handle the asynchronous invocation for long running jobs such as the computation services. This approach allows end users to submit their workflow specifications, input the parameters interactively with services, and disconnect from the system and retrieve all the data later.

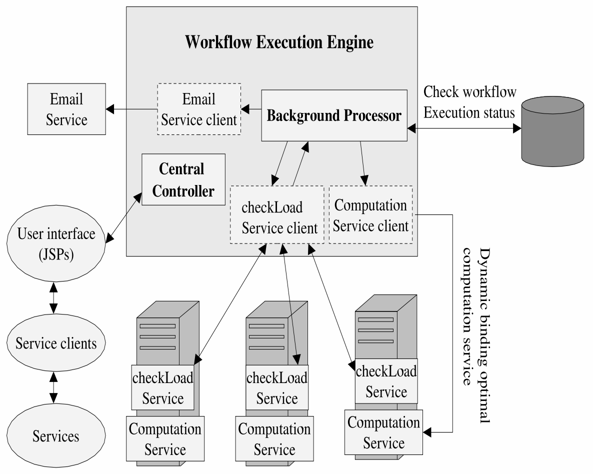

Figure 6. Workflow Execution Engine

The central controller is implemented by exploiting Java Servlets technologies. In our design, the end user visible web service hides the service invocation by providing user interface and service client. When the central controller begins to process the workflow, the controller processes the first job defined in the workflow by dispatching corresponding user interfaces to end users. By inputting the data, the service client invokes the web service and gets the results back. The user interface redirects the results to the central controller. If the message is an exception message, then the controller sends back an error message to the end user and informs the user that the task is terminated. If the message indicates that the job is done, then the controller saves the information in the session and dispatchs the next user interface to the end user. After all the sequential steps are finished, the servlet returns the end user a message to inform that all the information has been collected and the user can log out and check the results later. End users can check the status of their tasks from a JSP page. If end users want to terminate their tasks, they can send a request by clicking a button. In the mean time, the central controller sends a query to the database to indicate that the particular workflow is ready for the asynchronous execution.

The background processor is monitoring the status of all the submitted tasks by checking the database continuously at every time interval. When it discovers that a workflow is ready for invoking the computation service, the background processor first invokes the *checkLoad* service that resides on each server to get the CPU usage and memory usage at this time point and chooses an optimal computing resource. The background processor issues **(Runtime.getRuntime()).exec()** to invoke the computation service client that resides on the same machine. The computation service client dynamically binds the computation service on the selected optimal computing resource and sends a SOAP message to the computation service. The computation service client does not need to wait for the response from the computation service after it fires the call. After the computation service finishes the computation, it updates the status of the current task by sending an update query to the database that informing the background processor that the next service in the workflow is ready to be invoked. When the background processor gets this information, it invokes multiple data analysis services that are required in the workflow in parallel. The analysis results are stored in the database after the analysis processes are finished. After the workflow is finished, the background processor invokes the send email services to inform end users that their tasks are finished and the results are ready to be accessed.

7. Conclusions

We present a web services based portal architecture that supports end users composing new tasks without having to consider the detail description of web services. Service clients, provided by portal developers, reside between web services and end users. A service client acts as a proxy by hiding SOAP message sending and receiving. The portal architecture provides a simple specification that expresses complex dependencies between web services in XML. The workflow composer assists end users in combining a set of web services and generates a workflow. The workflow execution engine consists of a central controller that is responsible for synchronous service invocation and a background processor that is responsible for asynchronous service invocation. A mechanism is provided in the portal to choose an optimal computing resource for long running computation services and provide end users high quality services.

The workflow specification provided in the portal can be extended to handle more sophisticated control structure. It is also possible to incorporate semantic web technologies to provide more meaningful workflow vocabulary.

References

[1] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. Gridant: A client-controllable grid workflow system. In *Proceedings of the 37th Hawaii International Conference on System Science*, 2004.

[2] L. Arthurs, P. A. Maurice, X. Xiang, R. Kennedy, and G. R. Madey. Agent-based stochastic simulation of natural organic matter adsorption and mobility in soils. In *Eleventh Internation Symposium on Water-Rock Interaction*, June 2004.

[3] T. Berners-Lee, J. Hedler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[4] Bpws4j. http://www.alphaworks.ibm.com/tech/bpws4j.

[5] M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Model-driven specification of web services composition and integration with data-intensive web applications. *IEEE Data Engineering Bulletin*, 25(24):53–59, December 2002.

[6] Collaxa bpel server. http://www.collaxa.com/.

[7] gsoap. http://sourceforge.net/projects/gsoap2.

[8] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, pages 46–53, March/April 2001.

[9] M. Pierce, G. C. Fox, C. Youn, S. Mock, K. Mueller, and O. Balsoy. Interoperable web services for computational portals. In *Proceedings of High Performance Networking and Computing SC'02*, Baltimore, USA, 2002. IEEE.

[10] M. E. Pierce, C.-H. Youn, and G. Fox. The gateway computational web portal: Developing web services for high performance computing. In *International Conference on Computational Science (1) 2002*, pages 503–512, 2002.

[11] The globus project. http://www.globus.org.

[12] W. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, Jan/Feb 2003.

[13] Wsap. http://www.systinet.com/.

[14] X. Xiang, G. Madey, Y. Huang, and S. Cabaniss. Web portal and markup language for collaborative environmental research. In A. Scharl, editor, *Environmental Online Communication*. Springer, London, 2004.

[15] Xmethods. http://www.xmethods.net.

[16] J. Yang. Web service componentization. *Communication of the ACM*, October 2003.

[17] X. Yi and K. J. Kochut. Process composition of web services with complex conversation protocols: A colored petri nets based approach. In *Design, Analysis and Simulation of Distributed System DASD2004*, 2004.

[18] L. Zeng, B. Enatallah, and M. Dumas. Quality driven web services composition. In *WWW2003*, 2003.