Web Services Based Architecture in Computational Web Portals

By

Choonhan Youn

B.S. The University of Ulsan, 1991

M.S. Syracuse University, 1998

DISSERTATION

Submitted in partial fulfillment of the requirements for the
degree of Doctoral of Philosophy in Computer Science
in the Graduate School of Syracuse University
December 2003

Approved_____

Professor Geoffrey C. Fox

Date_____

Abstract

Computational web portals provide user environments that simplify access and integrate various distributed computational services for scientists working on particular classes of problems. The computational web portal, Gateway, consists of a dynamically generated and browser-based user interface that adds the client applications and a distributed component-based middle tier, WebFlow. The WebFlow middle tier provides a coarse-grained approach to accessing both stand-alone and grid-enabled back end computing resources. Like most computational web portals, Gateway was originally implemented in a three-tiered structure. This has inherent limitations for building portals that can easily interoperate and share services.

Specific application portals are typically built on common sets of core services, so reusability of these services is a key problem in Problem Solving Environment development. In this dissertation we address the reusability problem by presenting a comprehensive view of an interoperable portal architecture, beginning with a set of core services built using the Web services model and application metadata services that can be used to build science application front ends out of these core services, which in turn may be aggregated and managed through portlet containers. Managing multiple versions of services is an important consequence of this issue, and we close with a description of our work on negotiation for version control.

These portal services may be implemented in a programming-language and platform independent way using a Web services approach. However, security in Web services for distributed computing systems is an open issue involving multiple security mechanisms and competing standards. In this dissertation we present an implementation of a flexible, message-based security system that can be bound to multiple mechanism and multiple message formats.

We have developed QuakeSim portal for the earthquake science by presenting XML schemas and our design for related data services for describing faults, surface displacements, and specific simulation codes. These data services are implemented using a Web services approach and are incorporated in a portal architecture with other, general purpose services for application and file management. We then illustrate how these data models and services may be used to build distributed, interacting applications through data flow.

Table of Contents

List of Figures

List of Tables

Acknowledgements

During my research work, I should remember a few people to owe a great deal of gratitude. First of all, I would like to give my advisor, Dr. Geoffrey C. Fox my sincere appreciation for all of his challenging research objectives and invaluable advice. He has guided me through his continuous support and encouragement toward this research.

I would like to thank, my project leader, Dr. Marlon E. Pierce who helped me to have my work by providing a test bed, Gateway project which is based on Grid Computing environments and to identify those research problems with my approaches correctly, and to lead me in the right direction at this work.

I would like to thank the people at Community Grids Labs. for working closely with me and for providing lots of good information for experiments to me in this research.

I have some people who are worthy of a big gratitude for having served in my committee: Dr. Jongwoo Han, Dr. Stephen Chapin, Dr. Wojtek Furmanski, and Dr. Roman Markowski.

Finally, I would like to thank my wife, Deoghee, my precious daughter, Eunjin, my parents, and my parents-in-law for their deep love and commitment, and great affection. And I also could not have succeeded my work without their sacrifices.

Chapter 1

Introduction

1.1 Problem Statement and Contributions

Computational web portals are usually designed to provide coarse-grained grid approach that ties grid and non-grid computing resources and also simplify seamlessly access to computational grid technologies such as Globus[1][2][3], Legion[4][5][6], Condor[7][8], providing some portions of the infrastructure needed to support geographically ubiquitous, distributed computing and grid services such as high-throughput computing, metaqueuing jobs across multiple machines, high-performance file transfer, information management, and security, regardless of the location of users and high performance computing resources [9]. A general overview of the role of community technologies in computational grids has described in Ref [10]. Computational grid technologies are approaching maturity, but are still hard for users who are not

familiar with High Performance Computing (HPC) resources. These important issues are being addressed by grid computing environments, using web browser-based portals [11].

Web browser-based scientific portals also provide the user-centric view of computational grids. Building on a foundation of core services such as job submission and monitoring, file management, and session management, we can build sophisticated, domain-specific Problem Solving Environments (PSEs). Numerous such portals/PSEs have been developed, with varying degrees of specialization to applications. Some examples include NASA's Information Power Grid [12], San Diego Supercomputing Center's Hotpage [13] and its application-specific spin-offs, Pacific Northwest National Laboratory's Extensible Computational Chemistry Environment (Ecce) system [14], UNICORE [15], and our own Gateway project [16][17].

We have developed and are deploying a computational web portal system, Gateway [16][17][18], for the Aeronautical Systems Center (ASC) and Army Research Laboratory Major Shared Resource Centers(ARL MSRC), two high performance computing centers funded by the US Department of Defense's High Performance Computing Modernization Program. The important part of our design is devoted to offering an intuitive web browser-based interface that hides the difficulties and the user complexity of seamlessly accessing and using HPC resources in a secure manner.

The purpose of Gateway project is to provide web interfaces for a number of different applications including, for example, commercial structural mechanics codes such as ANSYS as well as in-house developed codes. Interfaces for these different areas can be extended from a set of generic services that provide the portal's core functionality. Those services include Job submission, Job monitoring, File transfer, Context (state) creation

and management, Queue script generation, Information services, Security, and Shared visualization [17].

Like Gateway system, most of computing portals are typically built around several basic services. These services may be built on top of Grid technologies such as Globus or SRB [19], but this is not always the case: the Gateway portal, for example, performs job submission by direct submittal to queuing systems, while Hotpage builds this service on top of Globus. The nature of web portals makes them appropriate for delivering both the aforementioned HPC-related services and more standard web-based tools (access to databases, collaboration tools, newsgroups, HTML documentation and help). In any case, the browser interface and the backend resources are separated by a middle tier that manages access to resources and communications, forming a three-tiered architecture. Each computational portal typically has its own code base for the middleware that can't be easily integrated with other groups' middleware. For instance, HotPage is built using GridPort, which is written in Perl, while Gateway uses WebFlow middleware, a custom Java/CORBA component system, and Mississippi Computational Web Portal(MCWP) [20] uses a Enterprise JavaBeans, a standard Java/CORBA/RMI component system and so forth.

A major shortcoming of the three-tiered computing portal design is its lack of interoperability. The three-tiered architecture results in a classic stove-pipe problem: user interfaces are locked into particular middle tiers, which in turn are locked into specific back end systems. An important problem that must be addressed by PSE developers or application managers is also the reusability and interoperability of their constituent core service components, based on the three-tiered computing portal. Obviously, application

developers want to reuse their own core service implementations to build new portals. We may go a step further and recognize the need for sharing reusable services between PSE development groups. Our experience has also shown that many of the basic services (such as batch script generation for queuing systems) are reinvented by many different groups [21]. A much improved process would be to build all portal services with well-defined interfaces and remote method invocation through a commonly accepted messaging system. Although consensus about common accepted definitions for interfaces, not to mention runtime interoperability, is hard to achieve between multiple groups, identification and reuse of another group's particular service tool from a common repository is a realistic goal, provided that agreement may be reached on how to plug these services into one's Problem Solving Environments. One powerful approach is to use XML, which provides a relatively simple, programming-language neutral approach, for common interface definitions and messaging to facilitate implementation independence. Web services [22][23] and the Open Grid Service Architecture [24] provide the specific XML languages for these mechanisms. With Web services, we now have a standards-based set of tools to properly build these XML wrappings and protocols. The crucial first step is to evaluate these technologies both for standalone and interoperating portals and to document these findings.

So, for next generation of computational portals, it would be useful for them to work together to develop along a Web Services model that will allow them interoperate. From the user's point of view, it is also useful for the user to achieve interoperable services among those computational portals, providing the critical contributions of this dissertation below:

- Portal developers don't have to reinvent every single important service.
- Users including researchers will have access to more services than any one project can provide.
- Users including researchers will be able to pick up the best available implementation of a service.

We believe that the key challenge to portal interoperability and reusability is to adopt lightweight, XML-based standards for remote resources accesses, service descriptions, and communication protocols. These are by their nature independent of the programming languages and software tools used by different groups to implement their clients and services using XML message-based standards such as SOAP [25], WSIL [26], UDDI [27], and WSDL [28]. We thus can build distributed object systems without relying on the universal use of heavyweight solutions such as CORBA [29], DCOM [30], or RMI [31], although these will certainly be included in the larger web services framework.

1.2 Organization of the Dissertation

The outline of this dissertation is as follows. Chapter 2 surveys the existing technologies related to our dissertation. Section 2.1 consists of the current portal components. Section 2.2 describes the current, major, Grid systems. Section 2.3 introduces the emerging standard Web Services components. In Section 2.4 the "portlet" technology is provided for the aggregating portal. Section 2.5 present the current portal systems developed and used by the other groups.

Chapter 3 presents the fundamental background related to our research, which includes the computational portal, Gateway system. Section 3.1 explains a traditional three-tiered architecture concerning with the computing portal, including user interface and distributed component-based middleware. Section 3.2 explains the portal metadata which represents application codes and resources needed for the portal user. In Section 3.3, portal services including job submission, file manipulation, session archiving, batch script generation, job monitoring, and visualization are explained, and in Section 3.4, one of important services, security service are described. In Section 3.5, we conclude Chapter 3.

Chapters 4, 5, 6, and 7 comprise the main parts of this dissertation. Chapter 4 presents works related to our methods and points out shortcomings of those works. Section 4.1 explains the problem overview related to portal services described in Chapter 3. In section 4.2 the comparison of CORBA and Web Services is investigated. Web service-based computing portal architecture is explained in Section 4.3. Section 4.4 introduces core Web services for computing portals related to interoperable and reusable services.

Chapter 5 introduces the Application Web services (AWS), which provides the scientific application metadata. Section 5.1 explains the general information and features about the AWS. AWS proxy component architecture is described in Section 5.2. In Section 5.3, we describe the AWS lifecycles which means four phases of the application existence during the interaction with the portal system. Section 5.4 explains how to represent applications, called AWS XML descriptors and schema design. In Section 5.5, AWS deployment and usage are explained.

In Chapter 6, Web service security and negotiation are proposed. In Section 6.1, the introduction about the security and the service compatibility is described, and in Section 6.2, we describe the secure Web services concerning with the security markup language, secure SOAP message format, message-level security infrastructure, and interaction with a Web service in a distributed environment. Section 6.3 presents the prototype of Web service negotiation as the quality of service, including the general overview, the motivating examples, Offer/Answer model, and implementation issues. Section 6.4 summarizes Chapter 6.

In Chapter 7, we apply our methods, which were introduced in Chapters 4, 5 and 6, to a real problem that is distributed earthquake modeling computing portal.

Finally, we summarize the contributions of our work and suggest plans for future work in Chapter 8.

Chapter 2

Survey of Technologies

In this chapter, we summarize a survey related to the field of Web portals, Web services, Grids with current technologies that motivated our work.

2.1 Web Portal Stacks

Portal services are the Grid components that control the marshalling of information from a variety of Web or Grid services and allow the user to view and interact with them. One needs to present the views of multiple Grid services in a way that is easy to customize for users and administrators.

The service-oriented architecture implies a component model for the middle tier and the underlying idea of modern portals is to match this with a component model for the user interface which will be built from “document fragments” with in the simplest case,

one fragment for each service. The portal then integrates or more precisely aggregates the individual fragments into a web page. Figure 2.1 shows this key portal stacks. We assume that all material presented to the user originates from a Web service which includes a content which come from a simulation, data repository or stream from an instrument, or a grid component. Each such Web Service has resource or service facing ports, which are those used to communicate with other services. We are also more concerned with the user-facing ports which produce content for the user and accept input from the client devices. These user-facing ports which are called Web Services for Remote Portlets (WSRP) use an extension of WSDL, which is being standardized by the OASIS organization.

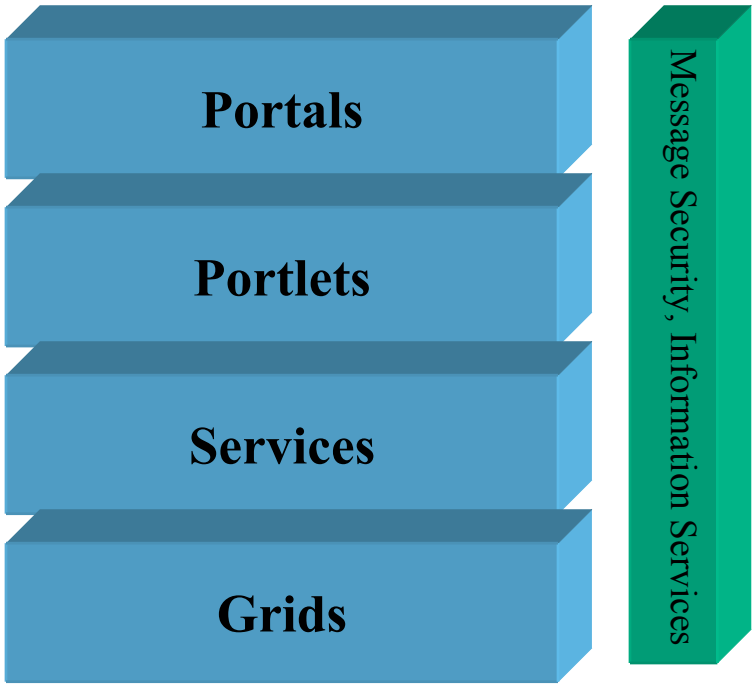


Figure 2.1: Web Portal Stacks

Most user interfaces need information from more than one content provider. One could integrate this in a custom application-specific Web service but it is attractive to provide a generic aggregation service. This allows the user and/or administrator to choose

which content providers to display and what portion of the display real estate they will occupy. In this model each content provider defines its own “user-facing document fragment” which is integrated by a portal. Such aggregating portals are provided by the major computer vendors and also by Apache in its well known Jetspeed project.

Portlets provide the desired component model for user interfaces in the same way that Web Services represent a middleware component model. Using this approach has obvious advantages of re-usability and modularity. One then has an elegant view with workflow integrating components (Web services representing nuggets) in the middle tier and aggregating portals integrating them for the user interface.

We survey current specific technologies for this each component from the Web portal stack at below sections.

2.2 Grids

The term “Grid” has emerged in the last decade to denote a proposed, integrated, distributed computing infrastructure for advanced science and engineering. The basic Grid concept is based on coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations [32]. Grid technologies and infrastructures support distributed virtual organizations with a mix of shared data and compute resources accessed both in asynchronous and synchronous mode. The emerging Grid infrastructure addresses these stringent and often conflicting goals by using a multi-tier architecture with a range of technologies. The current Grid infrastructure integrates ideas and capabilities from many areas. The best known Grid technology suite includes High Performance Computing activities such as Globus, Legion, Condor, NetSolve, and Ninf.

The **Globus Toolkit** [1] provides a commodity-based, open-architecture, and open-source set of services in the areas of resource location/allocation, communication, security, information discovery, data access management, fault detection, and portability for supporting Grids and Grid applications.

Globus Resource Allocation Manager (GRAM) services [33] provide a standard network-enabled interface to local resource management systems for grid tools and applications.

Communication services [34] are provided by the Nexus communication library which defines a relatively low-level communication API that is then used to support a range of higher-level programming models, including message passing, remote procedure call, remote I/O, and maintenance of shared state in collaborative environments.

Globus Metacomputing Directory Service (MDS) [35] provides a suite of tools and APIs for discovering, publishing, and accessing information about the structure and state of a grid as an information-rich environment in which information about system components is always available.

Globus Security Infrastructure (GSI) [36] has a modular design in which diverse global services are constructed on top of a simple local service that addresses issues of local heterogeneity. The local security service implements a security gateway that maps authenticated Globus credentials into locally recognized credentials at a particular site.

Legion [6] is comprehensive Grid software that enables organizations to collect a set of shared resources such as science and engineering applications and computing power and data to be used as a single virtual operating environment. It is object-based system composed of independent objects which are all things of interest to the system, for

example, files, applications, application interfaces, users, and groups. All Legion objects have a name, state, and metadata associated with their state and an interface as the metasystem. Core Legion objects provide mechanisms that allow user-level classes to implement chosen policies and algorithms as the Logion implementation of a Grid. Legion services are implemented naturally as objects.

Class objects manage particular instances. Managing instances involves creating them on demand, destroying them when required, managing their state and keeping track of them.

Host objects which are a machine's representative to Legion abstract processing resources. They are responsible for executing objects on the machine, protecting the machine from Legion users, protecting user objects from each other, reaping objects and reporting object exceptions.

Vaults objects are responsible for managing other Legion objects' persistent representations. They manage inert objects on persistent storage.

Implementation objects encapsulate Legion object executables and typically contain an executable code for a single platform and in general, any information necessary to instantiate an object on a particular host.

Implementation caches are used for storing implementation objects for later use.

Condor [8] is a high-throughput computing system and a specialized job and resource management system for compute-intensive jobs. It provides a job management mechanism, scheduling policies, priority scheme, resource monitoring, and resource management. The initial Condor concept forms the Condor pools which consist of agents, resources, and matchmakers together. Agents and resources independently advertise

information about themselves to a well-known matchmaker, which then informs the two parties that they are potentially compatible on a single machine. And then the agent contacts the resource and executes a job. Gateway flocking or direct flocking which pass information about participants between Condor pools is used for sharing resources across organizational boundaries, forming the worldwide Condor flock.

Condor uses matchmaking to bridge the gap between planning which is the acquisition of resources by users and scheduling which is the management of a resource by its owner as the nontrivial component of the grid computing.

Condor provides two problem solvers which means a high-level structure built on top of the Condor agent: Master-Worker (MW) and Directed Acyclic Graph Manager (DAGMan). MW is a system for solving a problem of indeterminate size on a large and unreliable workforce. DAGMan is a service for executing multiple jobs with dependencies in a declarative form.

Condor provides the standard and java universe that create a specific job environment. Standard universe is to reproduce the user's POSIX environment for a single process running at a remote site. Java universe is to create a complete Java environment for executing the user's job indirectly.

NetSolve [37] uses the remote computing paradigm and organize the three main components: the agent, the server, and the client.

NetSolve agent maintains a database of NetSolve servers along with their capabilities and dynamic usage statistics for use in scheduling decisions as the gateway to the NetSolve system.

NetSolve server is the computational backbone of the system and a daemon process that awaits client requests.

NetSolve client accesses the system through the use of simple and intuitive API. Using these APIs automatically contacts the NetSolve system through the agent, which in turn returns to the server, which can service the request, for example, running the job with the input data.

Ninf [38] is implementations of the GridRPC programming model, providing simple, powerful client-server-based framework for programming on the Grid, as with NetSolve system. GridRPC is a programming model based on client-server RPC, with features added to allow easy programming and maintenance of scientific application codes on the Grid. It consists of four components: client, server, remote executable, and information service.

2.3 Services

Web services describe an emerging XML-based distributed computing paradigm that differs from other approaches such as CORBA and Java RMI. Basic idea is to build a distributed invocation system out of existing Internet-based standards. Web services define the description of how to invoke service components to be accessed, wire protocol extension for conveying RPC calls, and the discovery mechanism for locating the service definition of relevant service providers. Web services are very loosely defined, when compared to other distributed computing systems and platform, programming language independent.

Web services standards are being defined by W3C and other standards bodies. We are particularly concerned with these three standards: WSDL, SOAP, and WSIL.

The Web Services Description Language (WSDL) [28] is an XML-based Interface Definition Language (IDL) for describing Web services as a set of endpoints operating on messages containing either document-oriented or RPC-style messaging. The service interface definitions including messages and data are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and the concrete representation of their messages for a diverse set of message formats or network protocols. Several standard bindings are defined the description of how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

The Simple Object Access Protocol (SOAP) [25] is a XML message protocol for exchanging structured, typed data in a decentralized, distributed environment. SOAP is a simple enveloping mechanism that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a remote procedure call (RPC) convention and responses. SOAP is the transport protocol independently so that SOAP payloads can be carried on HTTP, FTP, JMS, and so on.

The WS-Inspection Language (WSIL) [26] comprises an XML format for assisting in the inspection of a site for available services and a set of rules for how inspection related information should be made available for consumption. A WSIL provides a means for aggregating references to pre-existing service description published by a service provider.

A service description is usually URL to a WSDL document and can be a reference to an entry within a Universal Description, Discovery, and Integration (UDDI) [27] registries. The WS-Inspection document also contains a link to another WS-Inspection document. WS-Inspection documents may be created that allow the service requestor to pick and choose from the available descriptions and to make them network accessible.

Apache Axis [39] is a toolkit for converting Java applications into Web services and open-source codes written in Java provided by Apache group. We are concentrating on building Java Web services with it. Axis service deployment tools allow you to publish your service in a particular application server for example, Apache-Tomcat. Axis client tools allow you to convert WSDL into client stubs. Axis runtime tools accept incoming SOAP requests and redirect them to the appropriate service.

2.4 Portlets

A computational web portal may be built up out of user interfaces to the application interfaces for service implementation for applications as well as interfaces to core services such as file transfer or job monitoring service that a user may interest. Portlets/containers which are implemented in Java provide a simple way to do this.

The Portlet specification [40] will define a Portlet API that provides means for aggregating several content sources and applications front ends and is being standardized through the Java Community Process. The Grid Portal Consortium's initial architecture aggregates multiple services into a single portal using portlet containers. The container implements all portal specific services for managing user customizations, logins, and access controls and deals with all web content as generic ‘portlet’ objects for the controls

which portlets are displayed and how they are arranged. With portlets, we have a common infrastructure for managing the content and service user interfaces that are added in a well-defined way.

Technology already exists for aggregating these user interfaces. Jetspeed [41] is one popular, freely available, open source portlet implementation from the Apache Jakarta Project using Java and XML. It acts as the central hub where information from multiple sources is made available in an easy to use manner.

Portlets also can be defined as Web services. Web Services for Remote Portlets (WSRP) [42] services provided by OASIS WSRP TC define a web service interface for accessing and interacting with visual, interactive, presentation-oriented web services as a portlet. This WSRP functionality includes the *Producer* which provides portlets as presentation-oriented web services that can be used by aggregation engines, and the *Consumer* which consumes presentation-oriented web services provided by portal or non-portal content providers and integrates them into a portal framework. These web services for a portlet are built on standard technologies, including SSL/TLS, URI/URL, WSDL, and SOAP.

2.5 Portals

Some portals offer “seamless access” to a variety of computers. One example of this is Globus toolkit which offers the powerful concept of a common interface to multiple machines through the GRAM controlled by metadata with the MDS service. GRAM interacts with Condor in distinct fashions. Condor-G [8] can run “underneath” Globus and allows the GRAM to submit jobs to pools of Computers managed by Condor. As

well as technologies to provide an access for multiple computers through a single interface, one must be able to submit and manage the job submission itself to individual systems. There are many portals that control specific computers and of course the basic Grid technologies such as Condor and Globus offer this capability.

One would often build the overall management system of the execution of a job in many portal systems that would use Condor, Globus, or Legion at lower levels. Systems like the Java CoG kit, the GPDK development kit or GridPort kit are used to build the management system. The other example is the Gateway [16][17] or Mississippi portals [20], whose sophisticated management capability interfaces in this fashion. In particular, it is this level that must present a service (WSDL) view to users or other services.

The **Java Commodity Grid (CoG) kit** [43] provides middleware for accessing Grid functionality from the Java framework. It integrates Java and Grid components and services within one toolkit, as a bag of services and components and contains a set of command-line scripts that provide convenient access to Globus Toolkit-enabled production Grid from the client.

The **Grid Portal Development Kit (GPDK)** [44] provides Grid enabling middleware for the middle tier and aids in providing a Grid enabled application server. The GPDK is composed of three core components, Portal Engine (PE), Application Logic, and Presentation that maps to Model-View-Controller paradigm. The PE provides the control and central organization of the GPDK portal in the form of a Java Servlet that forwards control to the Action Page Objects which perform various portal operations, and the View Pages which provide a user and application specific display such as HTML form for invoking Grid services via the Java CoG kit.

The **Legion Grid Portal** [45] is a Grid computing environment project to make the Legion infrastructure for managing a Grid accessible to users via easy-to-use interfaces. It consists of six main components, the general portal mechanism which is a Perl CGI script, the portal interface which is the portal action pages, the session state which is caches, session files, and logs accessed by the portal, the legacy system which is a commodity database along with the scripts necessary to access it, specific portals which are used to run specific applications from the portal, the Grid infrastructure which refers to the services and tools provided in this case by Legion.

The **Mississippi Computational Web Portal (MCWP)** [20] is a framework designed for constructing application domain specific portals for simulating Distributed Marine Environment Forecast System (DMEFS) and Seismic Performance for Urban Regions (SPUR) project. It is realized as a modern multi-tier system. The front end is a Web-based interface for specifying the computational problem, allocating the resources, monitoring progress, and analyzing results. The middle tier split into Web tier (Web server) which accepts user requests and generates responses through interactions with the application server, and EJB tier (Application Server) which interacts with the back-end systems through the Java CoG kit.

There are important classes of Grid and web services to and from which information is streamed on a continuous basis. This whole area can be called "information aggregation" web services which accumulate the job status and related data. One example of this area is best exemplified by the well known HotPage technology from SDSC.

NPACI HotPage [13] is designed to be a single point of access to all Grid resources that are represented by the portal for both informational and interactive services and has

an unique identity within the Grid community. It is built using the Grid Portal Toolkit (GripPort). GridPort is a portal services layer that mediates between the backend and the portal layer that connects to distributed resources via Grid technologies such as Globus. HotPage is used to display the job status on HPC resources; this has been incorporated in the more advanced NPACI GridPort system from the University of Texas. The latter has enhanced HotPage to the more powerful GridPort Information Repository (GPIR) [46], which can handle job, NWS (Network Weather Service), GMS (Grid Monitor Service), and MDS data.

Chapter 3

Gateway System

In this chapter, we present one of the traditional three-tiered architectures, Gateway system, which is a commodity-based web computational portal that provides seamlessly a secure and uniform access to a general backend of compute, communication, and information resources. The Gateway computational web portal consists of a dynamically generated and browser-based user interface that adds the client applications and a distributed component-based middle tier, WebFlow. The WebFlow middle tier provides a coarse-grained approach to accessing both stand-alone and grid-enabled back end computing resources.

This Gateway system described below is the early stage of the Gateway web portal as the technical background. We did several projects for this system, including Gateway User interface, application metadata, security, and shared remote visualization. Based on

this background, Gateway system can be defined as another test bed with the emerging Web services model.

3.1 Classic Three-tiered Architecture

The objectives of Gateway are to provide seamless and secure access to high performance computational resources through the web-based interfaces. Gateway is implemented as a modern three-tiered system as shown Figure 3.1 [17]. Tier 1 is a high-level and browser-based front end for run-time visualization and particular custom applications built on the top of the web and object-oriented commodity standards. Tier 2 forms the distributed object-based and Object broker middleware. High Perfromance Computing (HPC) back end services comprise Tier 3.

The user establishes interactions with the remote servers through either a web browser or a client application, or both, using the over-the-wire connection protocols such as HTTP(S), IIOP [29], and SECIOP [47]. The middle tier consists of two basic parts, a web server running a servlet engine (a Java Virtual Machine, or JVM) and a distributed COBRA-based middle tier, called WebFlow. Gateway has JavaBean components for implementing specific local services. These can also act as proxies for WebFlow distributed components that are running in different machines. WebFlow servers comprise a master server and a mesh of child servers. The master server acts as a gatekeeper and manages all proxies and the life cycle of the children. These child servers can in turn provide access to remote back end services such as HPC resources running PBS or LSF queuing systems, visualization toolkits, a Globus grid, and the data storage devices.

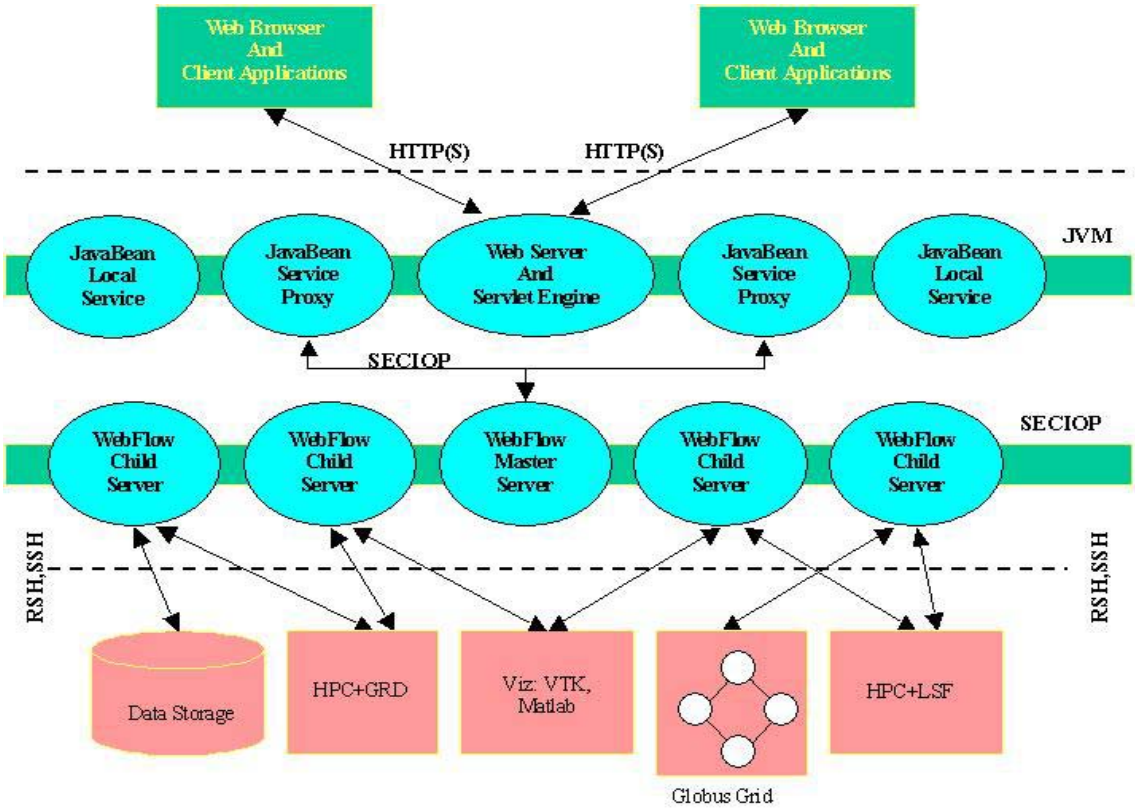


Figure 3.1: Gateway Portal Architecture

3.1.1 Gateway User Interface

The user accesses the Gateway system through the portal web page from the web server. One of the main functions of a WebFlow server is managing Gateway sessions using ContextManager service as described section 3.3.3. A session is established automatically after the user is connected to the WebFlow server by creating a user context. The user context is a container object that stores the user applications. The Gateway user interface organizes a user session into problem contexts inside a user context. Each problem context is then subdivided into session contexts. The data gathered from a particular session is represented internally as a linked set of hash tables called context data. This directory structure on the server is mapped into components in the

WebFlow middleware container hierarchy described below. This structure and context data are stored persistently for interacting with users.

In a simple reference implementation of the interface, we provide four tracks: code selection page, problem archive page, data archive page, and administration page.

- **Code selection page**: users who start a new problem domain make an initial request for HPC resources for running a particular application, and submit the job request script to the selected host's queuing system such as PBS, LSF. The WebFlow server generates and manages the problem and session context in a user context for storing the context data like the directory structure.
- **Problem archive page**: users are allowed to revisit and edit old problem sessions that were generated at Code Selection track so that they can resubmit their reconfiguring jobs to a different machine. Changed context data for a particular session context are stored in a newly generated session context.
- **Data archive page**: output data from old problems can be viewed with the visualization function from the problem archive page. It allows the user to visualize the data that is the generated output from his or her application, make an initial request for resources, and submit the job request to the visualization system like the code selection page. It also allows the user to display the image data that he or she has been visualized from a particular session.
- **Administration page**: privileged users and administrators are allowed to add applications and remote resources such as host computer to the portal XML application data descriptor, modify the properties of these entries, and verify their installation. This information is stored in an XML data record.

The Gateway user interface is developed using Java Server Pages (JSP) technology, including custom tags and JavaBeans, which allow us to generate dynamic, context sensitive web content and interface easily with our distributed Java-based middleware.

3.1.2 Distributed Component-Based Middleware

The Gateway middle tier consists of JavaBean components and a network of WebFlow servers. We partially implement the middle tier as JavaBean components. They act as the role of local services and proxies. However, JavaBean components all run on the same host server, which does not allow us to deploy a distributed middle tier on multiple, geographically distributed, hosts. These components that implement WebFlow Server API are easy to use for users who are not familiar with our WebFlow Server.

We have a distributed, object-based, CORBA-based implementation of the JavaBeans specification, WebFlow server for the computational portal. The merits of WebFlow middle tier have been described elsewhere [48][49][50]. In summary, it provides a network of proxies on machines that may have different file systems or that may even be operated by different organizations. WebFlow servers provide a hierarchical arrangement that is similar to the way in which LDAP directory servers and the CORBA naming service name their components. A WebFlow parent server acts as a gatekeeper to any number of child servers. Child servers can contain child servers of their own in this manner.

As depicted in Figure 3.2, the parent server creates and maintains proxy images of all its children and each component in a hierarchy. The primary goal of proxies is to forward requests from the web users to remote objects. Within a particular child, we can create

containers that map to the user's base context, problem contexts, and session contexts of the user interface. The WebFlow servers can also hold modules, which are CORBA implementation files that provide specific services such as job submission, file manipulation, session management, collaboration, and so on.

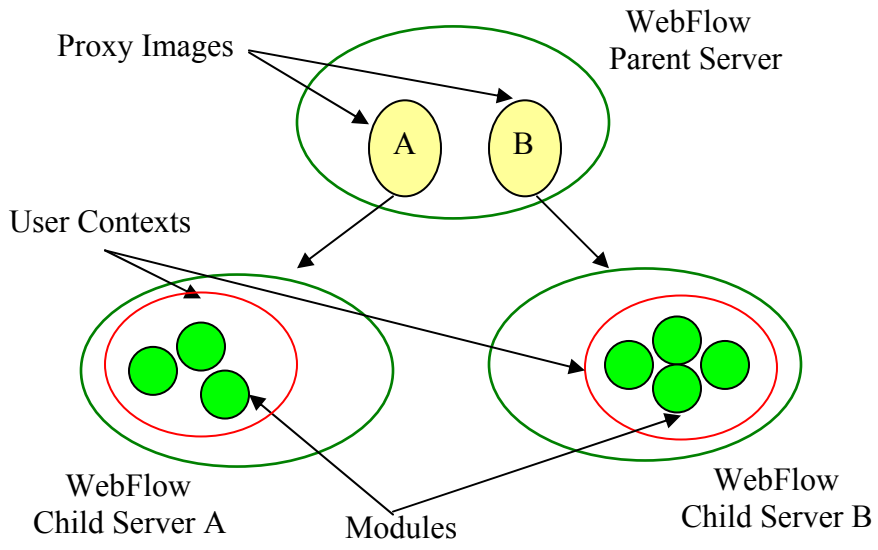


Figure 3.2: A container hierarchy of WebFlow Servers

Users contact and interact directly with the parent server only that maintains proxy images. We use a standard directory-style naming convention for child servers. For example, suppose the parent server name is "parent", a child server name is "child", and a grandchild server named "gchild" would be accessed through the proxy image of the parent by the name "parent/child/gchild". A parent server can contain the proxy images of any number of children and modules, but children have a single parent, resulting in a tree structure. A server within the hierarchy tree can contain both child servers and modules, and a particular module can be added to any number of servers. Thus, for example, two child servers at two different sites can each contain a module for accessing

the remote file system. A user connecting to the parent server can then look up each of these children and thus have transparent access to two file systems through a single entry point.

3.2 Gateway Portal Metadata

We can identify several groups of portal data that need to be described using XML. We refer to these objects as descriptors and have so far defined descriptions of Application description, HPC description, and Service description.

XML metadata descriptors are used to describe data that should remain long lived, or static and serve as the interface to host machines, applications, and available services, describing what codes and machines are available to the user, how they are accessed and used, and what services can be used to interact with them. We store this information in a static data record on the web server and use it to both generate dynamic content for the user and to generate back end requests.

- **Application Description**: By application, we refer specifically to third party scientific and engineering codes. A particular application descriptor contains the information in an XML data record needed to run a particular code. It would include the number of input files, the number of input parameters, the number of output files, the input/output style, and available host machines. By input and output files, we refer specifically to data files. Parameters are anything else that needs to be passed to the code on the command line, such as the number of nodes to use in a parallel application. This is highly code specific. I/O style includes standard Unix redirects or C-style command line arguments. It is not possible to

anticipate the requirements of every code, so this metadata should be easily extendible.

- **HPC Description**: HPC systems where codes run will also need to be described. A Host Descriptor would include the DNS name of the host, the type of queuing system, the full path of the executable on the host, the working or scratch directory on the host, the full path to the queue submission executable.
- **Service Description**: In addition to application and host descriptors, services also need to be described. This should be XML-based but will need to be translated into CORBA IDL to work with WebFlow.

3.3 Portal services

Gateway web portal has the generic services for Gateway system. It includes job submission, file transfer manipulation, context (state) creation and management, batch script generation, job monitoring, shared remote visualization, security.

We now describe how Gateway system is used to implement the basic services.

3.3.1 Job submission

Users want to submit jobs which are scripts including the queuing system such as PBS, LSF to HPC resources, or remote visualization jobs to the commercial visualization application tools. For this, WebFlow provides this service for executing submission requests either as local commands or as remote procedures using the rsh and ssh commands. We implemented these commands using Java external process. A child server

running directly on the desired HPC system and visualization system can perform the execution of the “local” command, because WebFlow servers can run on distributed hosts. Using the “remote” command, we can connect to another remote host from the WebFlow server.

3.3.2 File transfer manipulation

Users want to send and receive files and directories from their desktop to WebFlow server and from WebFlow server to their desktop back and forth.

A file transfer service allows directories and files to be transferred between the desktop and the remote servers using a simple user interface, supplementing file uploading and downloading through the browser. Although multiple file uploads are allowed in the HTTP specification, most browsers do not support this feature. So, we have implemented this service as a client-side application that interacts directly from the user’s desktop to a WebFlow server running a specially written module that can read to and write from the remote file system. Files are translated into binary arrays and then passed over the wire using either IIOP or SECIOP. We can use this service to send and receive entire directory structures, not including any subdirectories and files between machines.

3.3.3 Context (state) creation and management

We provide a session archival service for archiving interactions with the computational portal. Users can recover old state information and recover from a server crash within a user session. We refer to this service as ContextManager. This service

manages user, problem, and session contexts, which are mapped to a tree of directories on the WebFlow server. The directory structure is the user context, with subdirectories for problems and sub-directories for sessions. We refer to this data that are in these contexts as context data. Context data describes the parent context and child contexts, including any arbitrary name-value pairs in a particular user context. So, we store all information gained from the user's interaction with the browser forms. For example, the user's request to run a particular code on a certain host with a specified amount of memory, nodes, and wall time in a web browser form is represented as a series of name-value pairs that is stored in a directory that is associated with the problem and session contexts in which the request was created.

3.3.4 Batch script generation

Gateway provides a script generating service to aid users who are unfamiliar with HPC systems. This service assists users in creating job scripts to work with a particular queuing system such as PBS, LSF and so on. As a point of view of our experience, most queuing systems were quite similar and could be broken down into two parts, a queuing system-specific set of header lines, followed by a block of script instructions that were queue independent.

Queue scripts are generated on a particular WebFlow server, based on the user's choice of host machine, application code, input/output file name and parameter, etc. This information needed to generate the queue script is stored as context data, described Section 3.3.3. This allows the user to return to old sessions and revise portions of the

resource requests. If these requests are modified, then a new queue script can be generated using the modified context data.

3.3.5 Job monitoring

Job monitoring may be done through two basic mechanisms: server polling and backend events. Server polling means the server periodically queries the backend about the status of jobs. Back end events are the opposite. Here the backend server reports back to the server when the job reaches various stages (queued, running, finished, etc). There are some technologies for doing remote events such as Jini. Jini has the ability to handle persistent events, that is, a client can register as a listener for events, then disconnect from the system and reconnect later. While the client is disconnected, all of events are saved until the client logs back in. As using this mechanism, a client can submit a job that may take 48 hours or more on the queue. Clients are allowed to log off and log back in two days later to their output. So, we can use a Jini service running on the backend that notifies the client about various backend events such as job in queue, job running, and job finished.

We initially have investigated two basic ways of job monitoring: a simple event generating mechanism on the remote HPC resources, and server polling. The event system is described elsewhere [17]. Early experience suggests it is too hard to get working in practice. The polling system is currently used. We plan to investigate a more robust event system, which will also form the basis for workflow.

3.3.6 Shared Remote Visualization

Scientific visualization is the process of exploring, transforming, and viewing data and information as images to gain some deeper understanding and insight into data and hence promote greater understanding of what the data represents. The remote visualization also give researches and scientists tools for doing preliminary analysis remotely before beginning the time-consuming task of transferring large data sets to local machines for more intensive analysis. The general architecture and the considerations of the collaborative visualization are nicely described in Refs [51]. We are testing various schemes within Gateway for performing remote visualization and sharing these images between multiple users. In the web-based visualization, the traditional way of diffusing the images such as JPEG format and information over the web is that the publisher creates static web pages or image pages and links them through a URL. So, the publisher exports the sharing objects, for example, images such as JPEG image of the visualization results and then the viewer shares them as receiving the event that the publisher generates.

We describe the architecture of the shared visualization with three users, the publisher, the viewer A and B in Figure 3.3 [52]. The publisher is the master and the viewer A and B is using the shared display model to interact with the publisher. One important advantage of this model is its clarity of who is executing the visualization process.

An example of how remote collaboration can be implemented in Gateway using WebFlow is illustrated in Figure 3.1. This is similar to the general "Observer" design pattern [53]. First of all, we need the Collaborative Manager object that maintains all the connected users and the Collaborator object that describes each of the connected users for implementing the collaboration. The Collaborative Manager object maintains a dynamic list of all the connected users' Collaborator objects. It is also responsible for managing

the life cycle of the Collaborator objects. For example, when a user try to join a session, the Collaborative Manager object first checks to see if a user should be allowed access, and then creates a Collaborator object for the user and adds it to its list of listeners. When a user leaves, it removes the user's Collaborator object from the listener list. WebFlow also provides this life cycle management: this is analogous to what we do with the problem sessions with Gateway, that is, ContextManager described section 3.3.3. The Collaborator object is a WebFlow module that describes each of the connected users, such as permission, mode type and so on. It exports and replicates an object.

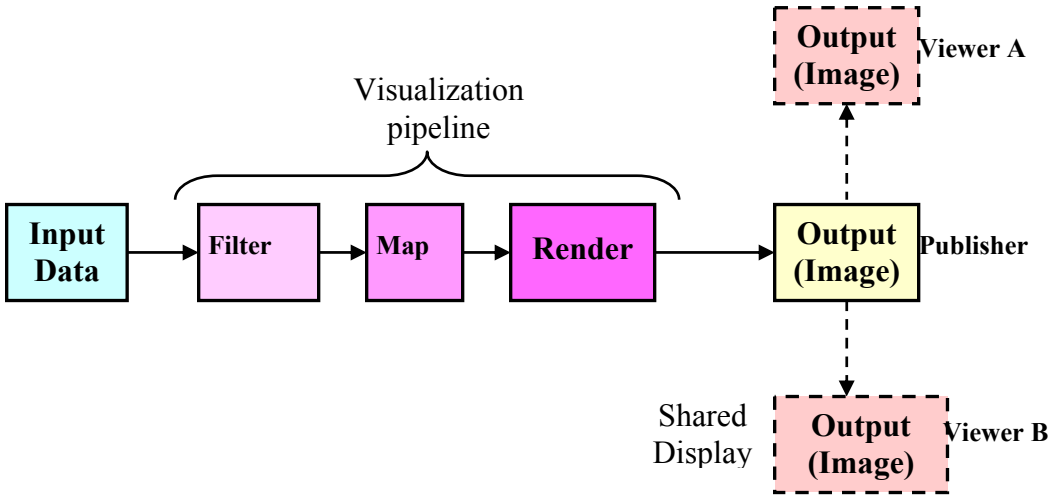


Figure 3.3: The architecture of the shared visualization showing the publisher and two collaborating viewers A and B.

The Collaborative user object acts as an interface to the Collaborative Manager object. Each user's Collaborator object has two control types, "master" and "slave". When a user becomes the publisher who has "master" control type and new data is available, the publisher generates event that all the connected viewers who has the "slave" control type listens. The publisher and viewer have the different Collaborator user objects. The publisher has the monitoring window that includes the mailing list of geographically

separated users connected through a network and the viewing window that displays the shared object. The viewer has the event listener window that listen event from the publisher and the viewing window that shows the shared object which the publisher exports. Refreshing the shared object, in the browsers can be done with a technique called client pull. It means that the web browser actually shows the contents from the first page and then waits some specified amount of time before displaying and retrieving the contents from the next page. Thus, client pull information is sent to the user using "Refresh" HTTP header. So, the viewer checks the event from the publisher using the client pull. Alternatively, we may use the technique called server push, in which the server and browser maintain and active socket connection. Updates are then pushed out from the server to the client as they happen. In testing we have found this method is not supported consistently by commercial web browsers, and when it is supported there are limits to the number of clients that can be maintained.

Simple shared visualization may be integrated into the portal using interfaces to such common tools as gnuplot [54] and MATLAB [55] as the scientific and mathematical computing, and visualization tools. Those are very popular tools for visualizing the scientific data. As described in section 3.3.4, batch script generation, likewise, the visualization scripts are generated on the server, based on the user's choice of the visualization system. This script is then moved to the selected remote host that includes the visualization system, for example, gnuplot, MATLAB using WebFlow module. The information needed to generate the visualization script is stored as context data, described Section 3.3.3.

3.4 Security requirements

There are no widely accepted standard security tools for a secure access to grid resources through the Internet. Computing centers apply different security technologies such as SSH, SSL, Kerberos, or other. Security solutions of commercial web sites are typically inadequate for computational grids and have no need to allow users direct access to its computational resources. None of these conditions apply to centers running computational portals.

Gateway security details are highly site-specific. The primary goal is to preserve the autonomy of the resources owner to define and implement its security policies.

3.4.1 Security in multi-layered architectures

Users, typically browsers, or custom applications for file browsing, contact a web server over secure HTTP connection.

Three-tiered web portals need at least two tiers of security, client-server and server-back end connection that must be authenticated, authorized, and made private. In addition, Gateway's distributed middle tier enforces secure interactions between distributed objects, including communications between parent and child WebFlow servers.

There are two basic mechanisms for security in a distributed computing environment: Public Key Infrastructure (PKI) [56] and Kerberos [57]. The Globus GSI [58] is an example of PKI-based security infrastructure for high performance computers. PKI (in the form of SSL plus server and client certificates) is also the security model for web.

Developing web portal applications around Kerberos presents many interesting challenges. Most importantly, no major browser supports it.

Gateway is being deployed in Department of Defense (DoD) computing centers that require Kerberos for authentication, data transmission integrity, and privacy. Centers also are required to use one-time passwords, created in effect by combining a static Kerberos password and a six-digit random passcode generated by a token card.

To minimize integration problems, we build the Gateway portal on top of the centers' preexisting security infrastructure. We now discuss how middle tier security among distributed WebFlow servers is achieved.

CORBA security service supports both GSS-API (such as Kerberos) and SSL [59]. Security features of CORBA are built directly into ORB and therefore they are very easy to use. We are using the secure ORB, which is implemented on top of the secret technology such as Kerberos, developed by Adiron Software, LLC [60] as the CORBA security service.

First, the client-server connection is the first layer of the portal that must be secured. This secure access is precluded simple deployment based on Kerberos users. The secure ORBs of WebFlow servers can be configured to authenticate using a preexisting Kerberos ticket-granting ticket (TGT). So, users should obtain their own Kerberos TGT through the Kerberos login. Client applications can thus prove authentication to the kerberized WebFlow master server.

Second, the distributed WebFlow servers represent the second layer that must be secured. The WebFlow servers typically do not run as privileged processes, whereas Kerberos makes the assumption that kerberized services are run as system level processes. Practically, kerberized WebFlow Servers has used a system's keytab file that has restricted access. For this, we have obtained special keytab files from the Kerberos

administrator that are owned by the application account and which are tied to a specific host machine. The user-to-user authentication would allow a client and server to both authenticate with TGTs, instead of a TGT and keytab. But, it is not typically supported by Kerberos implementations, although part of Kerberos specification.

WebFlow presents an additional problem to the kerberized ORB. The parent and child servers act as both client and server at different times, so the WebFlow server must possess access to a valid Kerberos TGT as well as the keytab. The internal representation of the Kerberos credential object in the WebFlow server is thus a composition of two credentials.

WebFlow servers may be run as a regular account. This allows us to deploy a single parent server that runs as a gatekeeper and multiple child servers, one for each user. This scheme presents a simple solution to authorization: the user's personal server has all the normal permissions and restrictions as any other process he or she runs. Users may then make requests to their associated child servers, but the requests are actually intercepted and invoked by the master server that acts as the user's proxy client and actually invokes the specific request. So, this requires two security features, mutual authentication and delegation policy. Mutual authentication is needed because the WebFlow child server, when it contacts the master to be dynamically added to the child list, must prove its identity to the parent. The secure ORB can be easily configured to use mutual authentication as a policy. Delegation is required because the initial client request is processed by the master, and so the client must delegate authority to the master to invoke a command on its behalf. It is possible to implement an additional safety feature in the gatekeeper that verifies the client has requested access to a child that it is authorized to

use. This can be done by inspecting the credential received from the client and comparing it to the name associated with the requested child server.

Finally, secure connections to the remote back end resources must also be made. To accomplish this, we create the user's personal server in the middle tier with a forwardable Kerberos ticket. The server can then contact the remote back end by simply invoking an external mechanism such as a kerberized remote shell invocation.

3.4.2 Kerberos security for web application

One of the difficulties of using Kerberos as an authentication service for web application is that it is not compatible with existing browsers. We now discuss two possible solutions to this problem.

One possible solution is to tunnel http requests through a secure CORBA connection. We call this service Charon, and it consists of a client-side proxy application and a WebFlow module that intercept HTTP requests and responses between the user's browser and an Apache server. This is illustrated in Figure 3.4.

A user wishing to connect to the Gateway web site must first get a TGT and then launch the client Charon application. He can then point his web browser to a specific port on his local host. All traffic through this port is forwarded to the Charon server (a module running in the WebFlow master server), which forwards the request to the local web sever. The response is collected by the Charon server and sent back to the Charon client, which sends it to the local port. All web traffic uses DES encryption and MD5 hashing to preserve message integrity. There are standard features of SECIOP. A drawback to the previous scheme is that it requires some installation on the user's desktop.

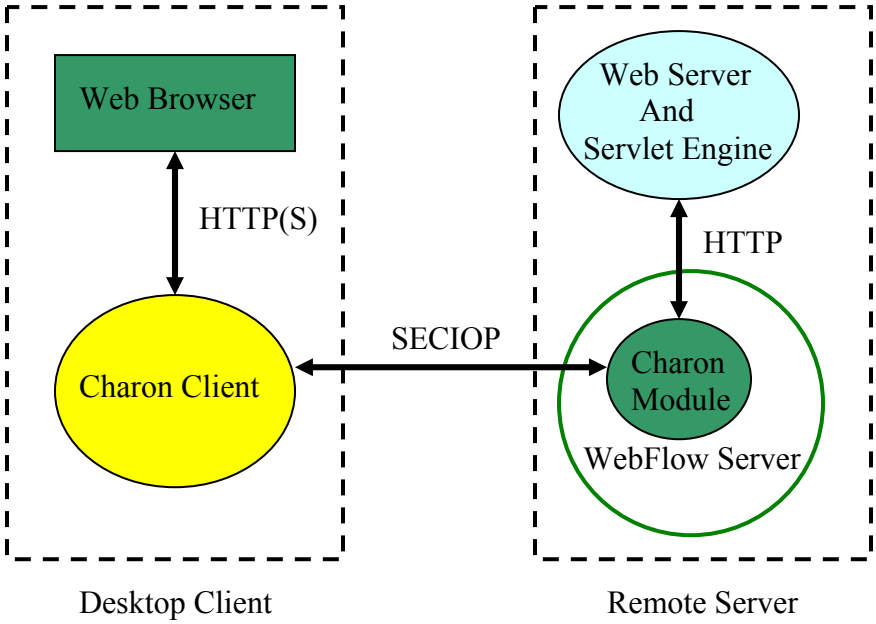


Figure 3.4: The Charon module is used to tunnel HTTP requests over a secure CORBA wire protocol.

An alternative solution is to provide a browser interface that will create a Kerberos TGT on the server for the user. The user would be authenticated in the usual way, with password and passcode. After the ticket is created on the server, we can use a message digesting algorithm to create a unique cookie for the browser. This cookie and the client browser's IP address are used for future identification. Before the browser can view secured pages, these two identification pieces are verified against the session values stored on the server. All wire communications should go over 128-bit encrypted SSL connections. Client certificates can be used as a third means of identification. The user can delete his server-side session tickets through the browser, and tickets should also be automatically deleted when the JSP session context expires.

3.5 Conclusion

We deployed the basic Gateway portal around the classic three-tiered architecture. Based on the distributed component-based CORBA middleware, WebFlow, we designed and implemented all of the basic Gateway services, security, XML metadata descriptors, and shared visualization service, needed for the computing portal. We also provided the Gateway user interface such as code selection page, problem archive page, data archive page, and administrative page which contains all of Gateway services that is described above.

Let us now examine the issues of portal interoperability from Gateway's perspective. For example, suppose that Gateway is deployed independently at Portal Group A and B. In that case, we have two problems: what is the best way to develop interoperable services between Portal Group A and B within Gateway, and what is the best way to achieve interoperable services between Gateway and Portal X deployed at another Portal Group C. This has inherent limitations for building portals that can easily interoperate and share services. Therefore, we will need another service-oriented based architecture on the emerging standards in the so-called Web Services approach. We will provide WSDL interfaces for all of Gateway services. In order for user to find Gateway services, we will publish those services into UDDI service registry. All remote method calls will use SOAP over HTTP.

Chapter 4

Web services based Architecture and Core services

4.1 Overview of problem

The traditional portal approach is concerned with accessing heterogeneous back end resources through a particular middle tier introduced in Chapter 3. Therefore, most portal projects are not interoperable as they are tied to their own particular interfaces of the middle tier. For example, the WebFlow middle tier, implemented in CORBA, can only interact with other objects that communicate with IIOP. Each portal typically has its own code base for middleware that can't be easily integrated with other groups' middleware as shown in Figure 4.1. For this, we can take our cue from the design goals of the underlying grid software [32]. In a network environment, interoperability means common protocols to which portal groups need to agree. Once this is done, they don't need to

make further agreements about the control code APIs and service implementations so long as the protocols are supported.

In order for portals to be interoperable, the below following things are needed:

- We need a common, simple wire protocol such as SOAP [25] that most portals can agree to use and that can be delivered using standard wire protocols such as HTTP.
- We need a common format such as WSDL [28] for describing services provided.
- We need to be able to obtain information about which service providers have what kinds of services. For example, we can use a service registry such as WSIL [26] and UDDI [27].

So, portal interoperability is desirable as an aspect of both the viewpoint of good software design and the realities of deployment.

Many different groups have undertaken portal projects [13][17][20][45]. It is in their interest to work together in order to take advantage of each project's particular strengths and avoid duplication of effort.

We consider the importance of the portal interoperability in two aspects. First, there are difficulties such as technical and political problems of getting a portal installed at a particular site. In order to deploy the portal system, we must satisfy the requirements of the hosting computing centers such as installing, testing, customizing security features, and so on. Most portals provide the general portal services such as job submission and secure access. Thus, once one portal has successfully been deployed at the computing center, it is redundant and time consuming for other portals to do the same if they can

instead make use of the security and job submission capabilities of the portal already in place. We call it an aspect of a site-specific grid computing web service.

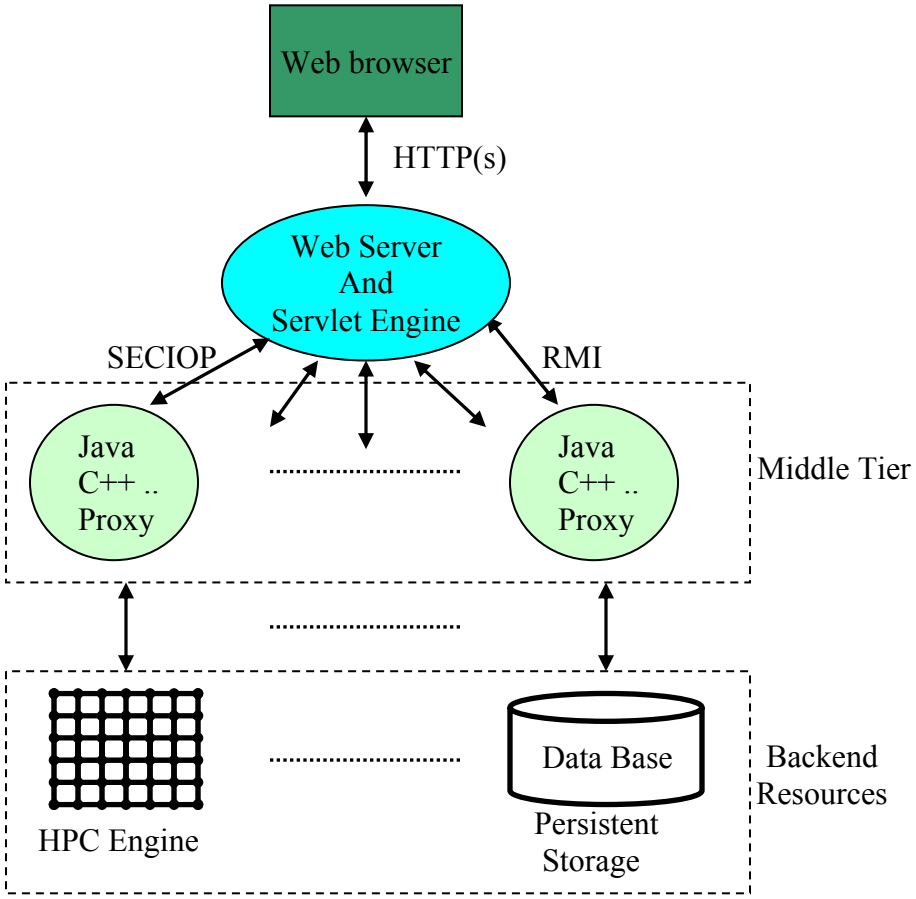


Figure 4.1: Each portal has its own code base for the middleware that can't be easily integrated with other groups' middleware.

Next, portals aid users to make the job queue script of multiple queuing systems on HPC resources. Gateway's proposed solution is pluggable only into portals with a Java-based infrastructure. The solution is to make queue script generation into a service that can be accessed through a standard protocol. A portal can then access the web service, provide it with the necessary information, and retrieve the generated script for its own use. This is an example of a function-specific grid computing web service.

4.2 CORBA and Web Services

4.2.1 The CORBA versus Web Services approach

CORBA has many of the same ideas as Web Services. SOAP is protocol part of the RPC mechanism of Web Services. The corresponding part of CORBA is IIOP. WSDL is the service interface specification of Web Services, and IDL is the corresponding part of CORBA. As the part of the directory service, WSIL, or UDDI of Web Services is the corresponding part of the Interface Repository (IR) of the CORBA.

While it is true that the OMG made a heroic attempt at language-independent API for distributed objects which is CORBA, it appears that it will not ultimately become the exclusive standard. On the contrary, many big players such as IBM [61] and Microsoft [62] actively support the Web Services approach. It is also very attractive to reuse the big infrastructure already built for the web and the HTTP protocol.

SOAP, which is endpoint independent, transport protocol independent, can use the simplicity of HTTP protocol or any other Internet protocol as the carrier for the ubiquity. So, it is bound to enter the domain that the IIOP has not. It has become generally recognized that the success of the Internet is due in part to the simplicity of the HTTP protocol. But, CORBA's IIOP has not penetrated the Internet as the HTTP has. For comparison, the specification for HTTP 1.1 is approximately 110 pages, compared to nearly 1000 pages for the core CORBA 2.2 specification. Likewise, the protocol is by its nature programming language independent, so different developers can produce their own API tools and software packages using the programming language of their choice, knowing only that the other end of the client-server connection speaks HTTP. For example, the Apache web server is written in C and the World Wide Web Consortium's

server reference implementation, Jigsaw, is written in Java, but both Netscape and Internet Explorer (IE) can access these web servers in a transparent fashion. Likewise, the server does not care if it is contacted by an IE browser or a web-enabled Java application. It only has to deal with the incoming request and assemble the appropriate response.

So, as based on only web-based client-server interactions, any two components on the web environments need to communicate. Instead of the powerful but heavyweight object sharing done by CORBA or RMI, SOAP is much more flexible to simply assume a simple common message protocol. If SOAP can run on the HTTP, SOAP messages can be routed through proxy servers and tunneled through firewalls by well established mechanisms.

Finally, we have learned some lessons from the CORBA. Actually, different vendors which implement the CORBA specification have the different ORB mechanisms so that they are not interoperable between ORBs in practice.

4.2.2 The performance test

We have developed a next generation computational environment built around interacting Web services. Each service can be thought of as a component in the software engineering sense. All components have web views that are easy to document while the universality of the web allows us to implement this model on essentially any distributed system. The W3C standard SOAP protocol is becoming very popular as a generic XML transport layer to be used when the performance is not critical.

So, a general characteristic of the Web services based architecture that we propose is that it supports an environment in which distributed components must interact via

communication channels using SOAP protocol at a reasonably coarse grained level. Components needing higher performance communication may negotiate down to a lower level protocol, but more typically we want to define the scope of our components by their communication requirements.

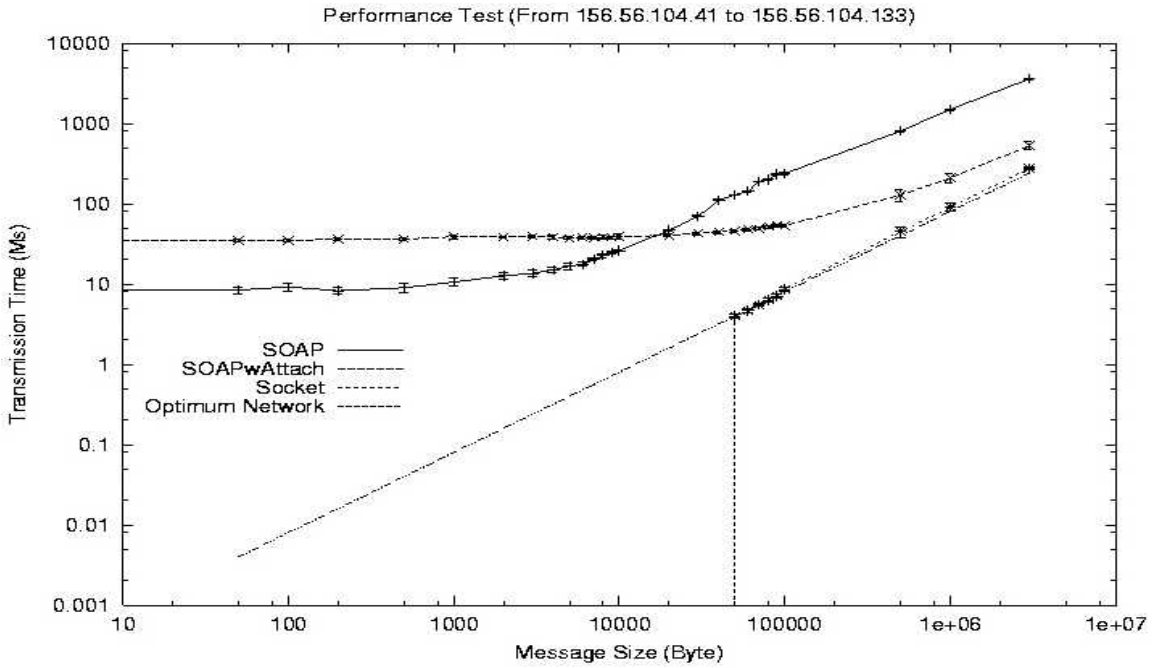


Figure 4.2: The measured transfer time for the message size on the same domain using SOAP, SOAP with attachment, plain socket and pure transmission time.

Figure 4.2 represents preliminary performance results for a specific service, file transfer. The plots show two different SOAP mechanisms for transferring text files: in the SOAP body and as a MIME attachment. For comparison, we also examined file transfer using direct socket connections. We compare this with the optimal network speed.

The bandwidth of Internet and Intranet connections is rapidly increasing but the latency of Web service component communication is likely to be in the 200 microsecond to one millisecond range. Actually, network speeds of SOAP messages may be in milliseconds range as shown in Figure 4.2. We have measured the transfer time for the

message size, so components should be chosen based in part on this communication speed. Typically this is not an issue for using our computing service components such as the job submission service component for executing the application codes, the job monitoring service component for checking the job's status, and so on. Likewise, components do not need microsecond response times like the message passing in parallel computing, either.

For testing the message performance, we classified four cases which we are interesting in a comparison as follows:

- SOAP over HTTP: For this case, we implemented a service which consists of one input parameter as the string type and one output parameter as the void type.
- SOAP with Attachments over HTTP: We uploaded a text file from the desktop client to the server using our "File Manipulation" Web service described in section 4.4.2.
- Plain Socket: In this case, we simply implemented a service that a client opens a server socket and then sends a text message to the server which just reads it.
- Optimum Transmission Time: We simply calculated the ttransmission time of each message based on the network speed, 100Mbps ideally for the comparision with other cases. The scale of inherent network latencies is assumed to be less than one millisecond in our tests.

As a testing environments, a desktop PC machine (domain name: 156.56.104.133) with Gnu/Linux 2.4.20-8 is used as the server host, and as the client host, we used Desktop PC (Window 2000, Intel Pentium 4 CPU 1700MHz, domain name: 156.56.104.41). These machines were connected by a 100 Mbps private network

connection. For both SOAP tests, we used Tomcat 4.0.4 and Apache Axis 1.0 to implement the services. Standard Java sockets were used for the socket test.

Measurements of transmission times were made in each of the four cases using 10,000 or more transmissions. The transmissions were grouped into samples of 100 measurements each and the average was taken. The mean transmission time was calculated by averaging the sample averages. The standard deviation of the samples was calculated in the usual manner.

The standard deviation assumes a normal distribution of transmission data: most of examples in a set of data are close to the average range, while relatively few examples tend to one extreme or the other. We found in the measurements that between 1% and 7% of the samples were more than 3 standard deviations from the total average. These large transmission times produced large standard deviations and were likely to do non-network causes such as unrelated operating system processes. We used a simple filter to remove outlying sample points: if a sample was more than 3 standard deviations from the total average, we discarded the sample, and the total average and standard deviations were recalculated with the reduced number of samples.

Network transmission times are normally assumed to increase linearly as the message size increases, with an initial offset, T_0, caused by network latencies. The transmission tests using pure sockets obeyed this simple rule, but the two SOAP tests did not. Pure SOAP messages show a zero-message size latency of approximately 8 milliseconds, and SOAP plus attachments have a zero-message latency of approximately 35 milliseconds. For this particular test, plain SOAP outperforms SOAP with attachments for messages smaller than 10 KB. For large messages sizes, SOAP with attachments appears to be

asymptotically approaching the optimal network speed, but pure SOAP messages appear to increase linearly (in log-log scale). Given these usual behaviors (non-linear dependence on message size, high latencies), we believe that the timing tests are dominated by less than optimal software implementations. We believe a more valuable comparison would be between different deployments of the same service using different Web service toolkits.

4.2.3 Shortcomings of Web services

CORBA specification has been begun since 1991, and since 1996 it has been a strong, useful, standard that produced fully capable ORB products from various vendors. Currently, CORBA has been evolved with most of the important pieces such as CORBA services and facilities. But, much more interest is currently going towards Web Services than other distributed computing technologies such as CORBA and Java RMI. Web Services/SOAP appeared around 2000 so that it is not available to the application developer as a well-known service and facility construct such as that of CORBA.

Therefore, Web service model itself is not provided service capabilities into the computing portal framework. We want to specify shortcomings of the Web services as follows:

- Primitive services needed for the computing portal system must be designed properly. That is, service interfaces must be simple for external users, and service implementation also must be self-contained for external users.
- Web Services extends to provide message-based security, or network-based protocol security such as SSL-based HTTPS.

- Web Services combines primitive services into useful application services which is metadata information services
- Web Services picture extends to include Portal and Portlets, for example, aggregation of Web services for scientific applications, aggregation of clients into components environments.
- Web services are not permanent or static so that it needs transient services to be created dynamically and registered. This is one of the issues being addressed by the Open Grid Service Architecture/Infrastructure [63].
- Web Services still should address numerous other issues, workflow such as Web Service Flow Language (WSFL) [64], quality of service for Web services negotiation, and as possible, the federation of services.

4.3 Web Service-Based Computing Portal Architecture

Most computing portals are based on a three-tired architecture and thus have a classic stove-pipe problem in aspects of services, as described in section 4.1. In order to integrate distributed services, the computing services should be designed for the interoperability and reusability. For addressing these challenges, we present a Web service based computing portal architecture around Web services model which have emerged as a popular standards-based, and service-oriented framework for accessing network-enabled applications.

Web services shown in Figure 4.3 have an XML-based, distributed computing paradigm to address the heterogeneous distributed computing services and the backing of companies such as IBM [61] and Microsoft [62] as the standard model for providing the

way of business-to-business and business-to-consumer services. It defines a technique for describing a service component, accessing a service, and discovering a service. Web services are loosely organized, and we do not imply that the exclusive use of all these languages or services defines Web services.

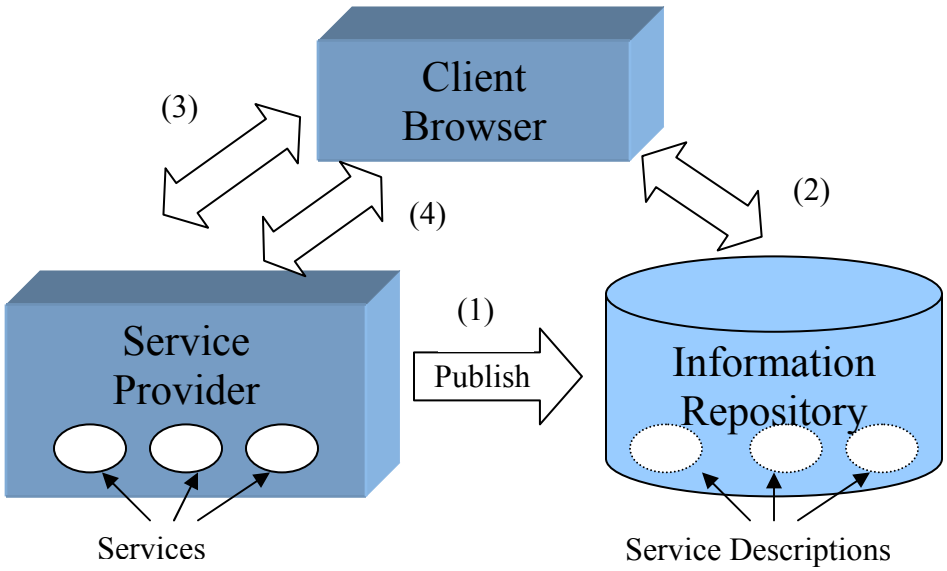


Figure 4.3: Web Services Model

For reference we provide the following brief summary of the major constituents of Web service systems. More extensive descriptions can be found in Ref. [22]. The steps are the following.

1. A service provider publishes its service to an information repository. An example of a service would be job submission. The publication is in WSDL, which describes what the service does, it's interface for using it, and how it can be accessed (i.e. this is a job submission service, it needs this kind of input, generates this kind of output, and you can reach it with SOAP, IIOP, or HTTP).
2. A client browser (such as a user on a web browser) wishes to use a service, so it contacts the information repository to find where such a service resides. The

information repository provides this information, telling the client everything it needs to know about contacting the server and using its service.

3. The client then uses this information to bind to the server.

4. Finally, the client uses the service.

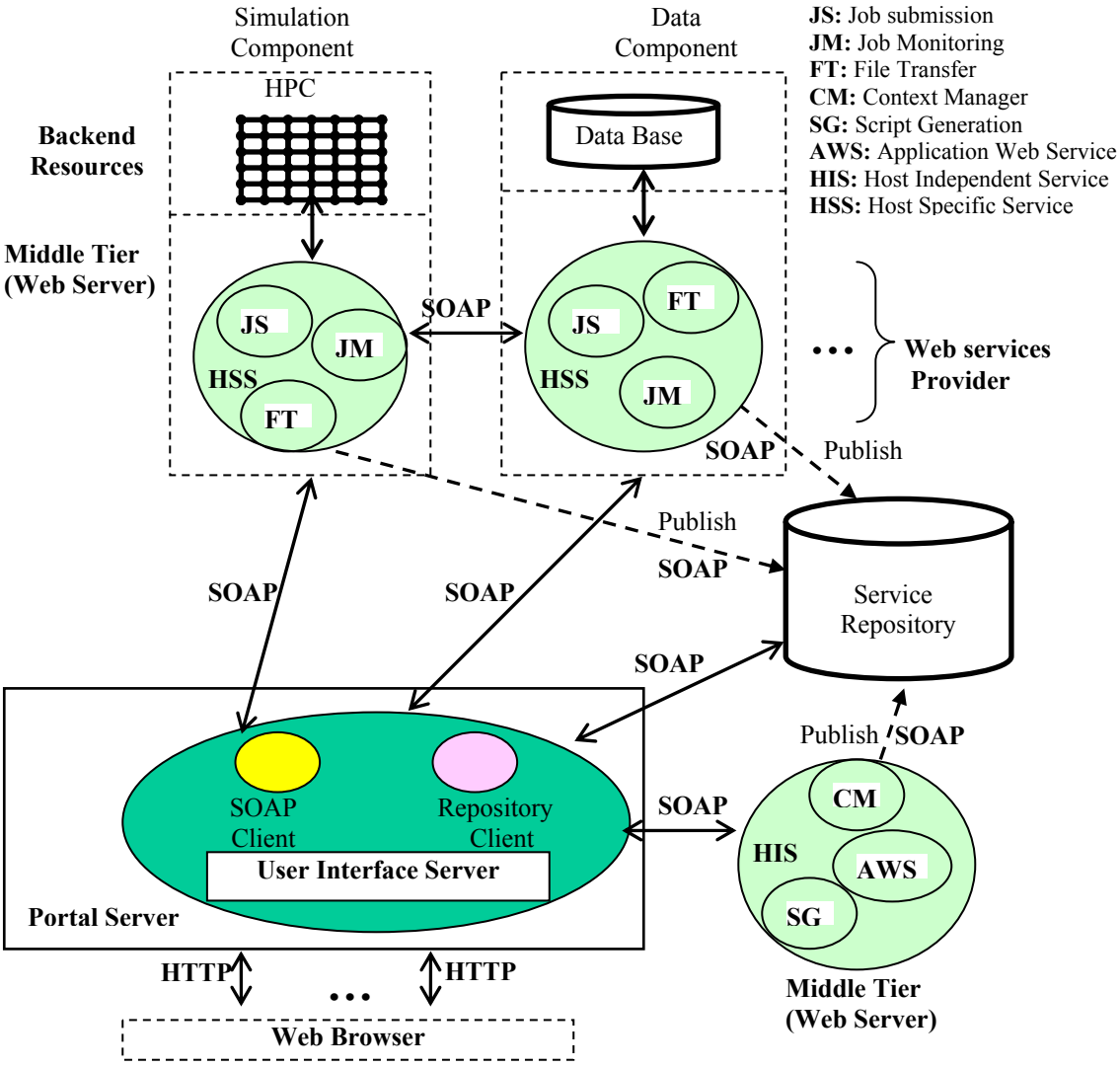


Figure 4.4: Architecture of Web service-based computing portals.

Let us now consider the aspects of this model for high performance computing.

- **Distributed object model:** The point of view that we take is that all back end resources should be considered as objects. There will be well-defined software

objects such as CORBA objects, but may also be hardware, applications, user descriptions, online documents, simulation results, and so on. As described below, these objects will be loosely coupled through protocols rather than particular APIs.

- **Resource and service description:** The generalized view of resources as objects requires that we describe the metadata associated with the object and provide a means of locating and using it. XML is appropriate for the first and we can assign a Uniform Resources Identifier (URI) for the latter. WSDL is appropriate descriptor language. This defines objects in a language independent way, allowing them to be cast into the appropriate language later, using for example Castor [65] to create Java objects. Likewise, this distributed object system is protocol-independent. The object or services must specify how it is accessed, and can provide multiple protocols.
- **Resource and service discovery:** Once we have described our object, it must be placed in an XML repository that can be searched by clients. Clients can thus find the resource they are looking for and how to access it. Commercial web services tend to use UDDI as service discovery, but other technologies such as LDAP servers may also be used. Castor provides a particularly powerful way of converting between Java components and XML documents and when combined with an XML database, can serve as a powerful object repository specialized for Java applications. For example, GXOS that is using as the XML description language in our Gateway Web Portal is developing by using Castor.

- **Service binding:** Following the discovery phase, the client must bind to the remote service. WSDL supports bindings to services using different mechanisms including SOAP and CORBA. Thereafter, interactions can be viewed as traditional client-server interactions. There is nothing particularly new about this architecture. Client-server remote procedure calls have been implemented in numerous ways, as has the service repository. As just one example, WebFlow implements a specialized naming service for looking up services and their service module. The difference here is that the repository is designed to offer a standard XML description of the services using WSDL. The service description itself is independent of wire protocols and Remote Procedure Call (RPC) mechanisms. There still is the problem that the client must implement the appropriate RPC stubs to access the server. But, given the number of protocols and RPC mechanisms available, there is an advantage in the various grid computing environment projects coming to an agreement on a specific RPC protocol. Guided by the surge of development in support of SOAP for web services and its compatibility with HTTP, we suggest this should be evaluated as the appropriate common language for interoperable portals. Legacy and alternative RPC mechanisms such as used by WebFlow can be reached through bridges.

The architecture for this kind of Web services system from the point of view of a portal is illustrated in Figure 4.4. The basic point is that common interfaces can be used to link different multi-tiered components together. In this figure we have two distinct backend hosts, a high performance computing host and a database host, perhaps implemented by different groups. These hosts run various services that interact with the

host to execute operating system command, etc. Information about these services is maintained by one or more information services. The user interacts with the service hosts and information servers indirectly, through client proxies maintained by the User Interface Server (UIS). The UIS is responsible for aggregating the various core services into application-specific Problem Solving Environments. The various component interfaces may be collected as *portlets,* as described in [21], which define how user interface components can be plugged in and managed by portal administrators and users.

Jetspeed [41], for example, is an open source portlet container system. Client stubs can bind and access these services with the protocol and mechanism prescribed in the service description by first contacting the service repository, UDDI that maintains links to the Web service Providers' WSDL files and server URLs and finding a service to use. In this architecture, the control layer between the server that manages the user interface and the server that manages a particular service becomes decoupled. This separation makes it possible to provide the interoperable (or at least pluggable) services.

4.4 Core Web services for Computing Portals

From Figure 4.4, we may classify services as either host-specific services (HSS) or host-independent services (HIS). HSS includes job submission, job monitoring, and file management service. Instances of these services are bound to particular hosts. HIS include context management, script generation, and application services. HIS services are usually information/metadata services that are not tied to specific service points; i.e. the service provided does not depend on the location. We consider the above to form a basic set of portal Web services. In general, these services must be chosen so that they define

properly course-grained functions with concise interface definitions. We now present several examples of such services.

4.4.1 Job submission

Computational portals must obviously allow users to execute scientific applications. We have defined a WSDL interface shown in the A.1 of the Appendix for executing commands on specific hosts systems. This service is remotely accessed through SOAP messages over HTTP. The service may execute operating system calls directly or may interact with Grid services through client APIs. We implement this service in Java and typically but typically use it to run external (non-Java) commands. We usually couple this service with the batch script service described below.

4.4.2 File Manipulation

Portal users must be able to move files between their desktops and various backend destinations, as illustrated in Figure 4.4. They must also be able to manipulate remote files transparently. We have defined Web services for such file management, allowing users to transparently move, rename, and copy files on remote back-ends. Files may also be transferred between desktop and backend, and cross-loaded between different backend sites. The full service interface may be obtained from the A.2 of the Appendix.

File uploading and downloading services illustrate the use of SOAP messages with attachments [66] in the RPC messaging style. SOAP attachments are non-XML files that are appended to the SOAP message and are useful for sending binary data and files with known MIME formats. For example, the file uploading service sends SOAP messages

with attachments encapsulated in a MIME multipart format from the desktop to the SOAP server. We implemented file uploading and downloading service using the DataHandler class from the JavaBeans Activation Framework [67]. This class is used to represent arbitrary binary data (which could include text files, binary documents, and program data files). This provides a consistent interface to data available in many different sources and formats. Apache Axis [39] provides the serializer and deserializer for the DataHandler class so that a SOAP client (in Figure 4.5) may send SOAP message with attachments to a remote SOAP server that is running the file management Web service. This service must still implement the file management in details described in the WSDL interface for the received file.

We further allow the SOAP client to instruct one service to move a file to another service directly. We refer to this as cross-loading. For example, in Figure 4.4, a user may request that a file be transferred from the data storage component to the simulation component. The file management service implementation has the capability of also acting as a SOAP client to another service. Here the data component service receives a cross-load message from the SOAP client and uses this to construct a new SOAP upload request to the simulation component.

4.4.3 Context Management

The Context Management service archives interactions with the computational portal. Each user is assigned a unique set of context data, which is used to store all information from the user session. The default context data contains “Date”, “LastTime”, “Descriptor”, “Directory”, and “CurrentChild”. In general, context data can be used to

store any useful metadata. For example, we define user session data using application instance metadata described in Chapter 5, but context data can be used to store the location of this XML file. Context data can later be recovered and edited by users to, for example, modify old sessions in order to resubmit jobs.

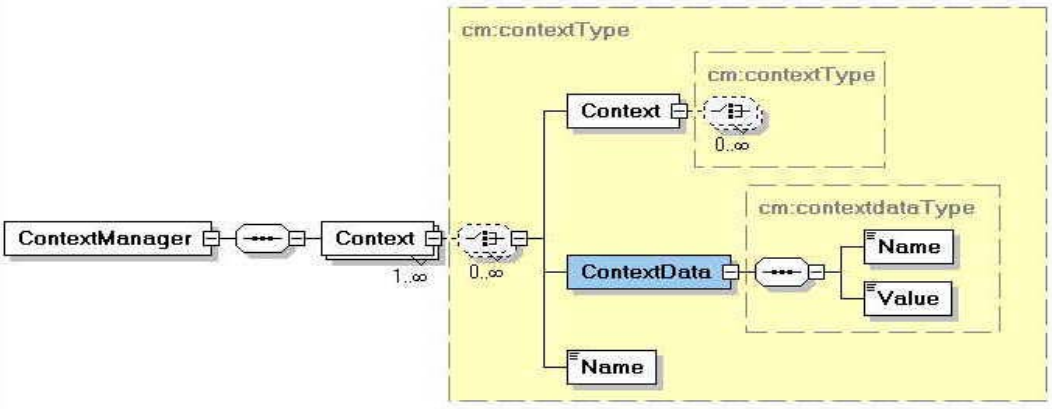


Figure 4.5: Schematic diagram for the Context Manager XML descriptor.

In our terminology, a context is a container that can hold an arbitrary number of string name-value pairs, as well as other contexts. These contexts are defined as a recursive XML schematic diagram shown in Figure 4.5 (see example given in the B.1 of the Appendix for more detail), so the schema supports an arbitrarily deep and complex tree-shaped data structure. In practice, a user's context data consists of a root context, with child contexts for particular problem groups for that user and grandchild contexts for particular problem sessions.

The context data service is built over the data model described above. We refer to the actual service interface for manipulating the context data as the Context Manager. This is defined in WSDL and exposes the following methods for manipulating context data:

- A user can add one or more contexts which is called the problem domain and sessions and so on. The context path should follow UNIX path style. For example,

the current path for a user context is empty character (“”). So, if a user has “test” context for the problem domain and want to add new context, “session” under that context, the context path should be used “test/session”. Using the JXPath [68] which implements XPath, the context path is checked whether or not the context is available.

- A user can search the context data which is stored in a user context during the portal interaction using XPath queries and store the context data which is the input from the portal interaction for editing and managing in a specified context using the context path.
- A user can remove the specified context including all of contexts and the context data in a user context using the context path.
- A user can get the list of “children” list of that context for reviewing the context data, using the context path.

The WSDL for the entire interface may be given in the A.4 of the Appendix. The above method calls are used for internally manipulating the context data. Internally, the context data instance is represented as a set of data classes created with Castor [65]. We store these schema instances persistently either on the file system (mapping context data nodes to directories) or in an XML-native database such as Xindice [69].

We described above how to implement a Context Manager Web service but not how to use it. Application Web Services (described Chapter 5) give the user various choices and are used to generate forms needed to collect the information needed to run the code. The user’s particular choices constitute a separate XML document, the Application Instance Descriptor. This contains all the metadata about a particular invocation of the

application. Using the Context Manager Web service, a user can add the session context in a user container structure that can be mapped to a directory structure for storing this user application instance XML file. So, a user can review and edit these data from the session context which contains all of information. From the user's Application Instance XML document, the user's job script which contains the queuing script such as PBS and the user script which the user run on the application code is created by the Script Generator Web service described in Section 4.4.4. This job script for a user is stored by Context Manager Web service which maintains the user's information. When a user submits the job, a user gets the location of the user's job script by the Context Manager Web service running on some particular host.

4.4.4 Script Generation

We have developed a Batch Script Generation (BSG) Web services for users who are unfamiliar with high performance computing systems and queue schedulers. The WSDL interface may be given in the A.5 of the Appendix. This service assists users in creating job scripts to work with a particular queuing system such as the Portable Batch System (PBS). From our experience, most queuing systems were quite similar and could be broken down into two parts, a queuing system-specific set of header lines, followed by a block of script instructions that were queue independent. Queue scripts are actually generated on the server, based on user's choice of machine, application code, memory requirement, input/output file name and parameter, etc. This information is stored as an XML document, the Application Instance Descriptor (described in Chapter 5). The BSG service accepts this XML document as an argument and generates the queue script of the

requested type. The structure of the BSG service implementation allows it to extensibly support different queuing systems.

A previous version of this service has been described in Ref [70]. We have extended this service to integrate it with the Application Instance schema described Chapter 5. The latest version returns both the generated batch queuing script and the shell script needed to submit the queue script to the queue of a specified host. All information needed to generate these scripts is obtained from the Application Instance data. An actual submission uses the Job Submission service described in section 4.4.1. Scripts may be moved to the appropriate host using the File Management service described in section 4.4.2.

4.4.5 Job Monitoring

Job monitoring services may be built in one of two ways: periodic client polling and server event notifications. We currently use the polling method, which we prefer for reliability, but have built event-style prototypes based on email notifications.

The polling job monitoring Web service makes one method available for use by clients through a SOAP RPC for monitoring the execution of a job running in a queuing system. The WSDL interface may be given in the A.3 of the Appendix. Basically, a batch job submission returns a unique job identifier that can be used for enquiry about the job status. If the job is submitted to a batch scheduler it is in the pending state while sitting in the queue waiting to be executed. The job is active when actually executing and may become suspended due to pre-emption mechanisms. In case of normal completion the job status is done, otherwise the job is failed.

The input to this method is the user account name and the scheduler type, such as "PBS". The service implementation is designed as a factory so that support for particular schedulers can be added in a well defined way. If the scheduler type is not supported by the Web service, then a message to the effect is returned to the client. If the scheduler is supported, then the user name is passed to a handler created for that specific scheduler. The scheduler handlers are custom-written methods that generate a WSDL complex type, effectively an XML data object given the user name and return the array of the generated a WSDL complex type that contains the job status of the scheduler.

4.5 Summary

In this chapter, we have presented the Web service-based computing architecture around the Web service model and also the design and implementation of several core portal services such as Job submission, File Manipulation, Context Management, Script Generation and Job Monitoring. These services form the basis of the computing portal environments, and we have developed interoperable and reusable services that can be used for other computing portals. We believe that the current infrastructure provides the service interoperability and reusability.

Chapter 5

Application Web Services (AWS)

5.1 Introduction

The core services described in Chapter 4 are intended to serve general purposes. These must be organized in a more meaningful fashion for use in computing portals. In particular, we have developed a set of XML schemas for describing scientific application metadata. Here "application" means specially a particular set of science or engineering codes, such as finite element codes, grid generation codes, and visualization tools which are developed by the science community on the computational grid. We will treat all these applications as black boxes and will describe these applications as XML Schema Descriptors which is applicable to the computing portal. For this, those schemas can be converted into Java data classes for the manipulation by services. As distinguished from SWIG [71], which is the direct wrapping for an application, using script language such as

Perl, Python, and Tcl/Tk, an actual application is wrapped by general purpose Java applications, which can be used to invoke the application, either directly or through submission to a queuing system without modifying the application source code. Thus, we define a general purpose set of schemas that describes how to use a particular application and bind it to the services it needs. These schemas are the foundation for what we call *Application Web Services (AWS).*

We have developed the AWS for providing the following features:

- We want to be able to describe applications in a general way that is independent of the particular portal. The application XML descriptors can be standard across portals.
- In addition to defining the application interface, application descriptors also specify the core services that are required to run the application and provide context in which those services are used.
- Having a general application description mechanism allows user interfaces to be developed independently of the service deployment.
- An additional general capability of the AWS is that the application metadata may be discovered and bound dynamically through XML database, Xindice [69] for storing and querying.

Based on those features, applications services are deployed into a computational web portal, a browser-based system that provides a user interface to applications and various services. A typical web portal allows a user to log in securely to some computing resource, submit jobs, view results, and manage files on the remote resources. The web

interface is desirable since it allows the user to potentially log in from anywhere with no software on the client besides the web browser.

We believe the process we introduce here for transforming applications into services is generally applicable to any portal designed to support multiple codes, particularly when the applications to be deployed are not known *a priori* by the portal developer. We now use Gateway as an example of deploying and using an AWS, although the process should be similar with other proxy-based portals.

5.2 AWS proxy component architecture

Figure 5.1 summarizes our general architecture. An actual application (a scientific code or a database, for example) is wrapped by a Java program. For example, in case of databases, this is well known: the Java application just makes a JDBC connection to the database and defines and implements an API for clients to interact with the database. Likewise, scientific applications can be wrapped by general purpose Java applications, which can be used to invoke the application, either directly or through submission to a queuing system.

The WSDL interface is the XML abstraction of this wrapper application interface and can be viewed as a list of instructions for building clients. The actual client (say, a JSP page) may be developed offline, or (as in the figure 5.1) generated automatically by mapping the application interface into visual components (like HTML form elements). These user interfaces to services further can be wrapped as *portlets* and aggregated into a single user interface using a technology such as Jetspeed [41]. Thus in the grand scheme towards which we are building, a Web portal consists of client user interfaces to various

services generated from the XML description of the service. Likewise, this XML description of the service describes how to invoke the service methods, which in turn are just proxies to some legacy application.

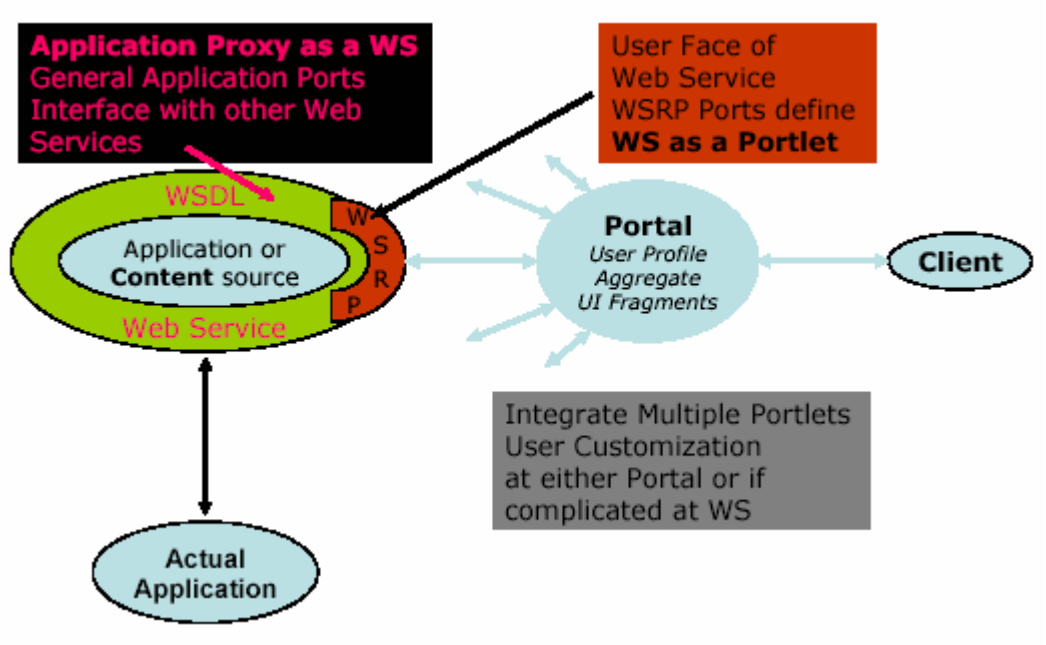


Figure 5.1: The proxy component architecture

5.3 AWS Lifecycles

Web Services for science applications have at least four phases of existence for deploying in computing portals as follows:

1. Application abstraction state: a general description of how to invoke the application (it takes 1 input file, generates 4 output files, lives on computers A, B, and C).
2. Prepared instance state: this is a specific instance of the application. A user has provided most or all of the information needed to run an application, but has not yet actually queued or submitted it.

3. Running instance state: This is job that has been committed to a set of resources (submitted to a queue, for example) for running the application. That is, submitting jobs means that jobs is submitted to a queuing system which represents the additional states, such as queued, running, sleeping, completed, and terminating.
4. Completed (Archived) instance state: The job running has finished and then we preserve the application metadata about the application.

All of this process can be described by a set of XML descriptions languages. Based on these classifications, we have defined the two sets of schema descriptors: the first set describes the abstract state of (1) above about a user's information about applications and the second set describes the application instances in states (2)-(4) above about a user's actual choices.

Operationally, an instance of the abstract application schema associated with state (1) above is edited by the application manager to describe how to invoke his or her application. This description is then viewed by application users, whose specific choices are instances of application instance schema associated with states (2)-(4) above.

5.4 AWS XML Descriptors

The abstract application description is implemented as a set of three schemas: application, host, and queue. These are implemented in a container hierarchy. So, the abstract application XML descriptor contains one or more applications which contain one or more host XML descriptors, which in turn contain one or more queue XML descriptors. Specially, as we chose this modular approach, host XML descriptor and

queue XML descriptor are highly adaptive and pluggable with other best schema which is applicable to the application descriptor.

We continue to refine our application description schema as we implement new features (such as core service bindings) and apply our descriptors to more applications. The current version of the schema shown in Figure 5.2 is available from the B.2 of the Appendix for more information in detail and has the following essential elements:

- A base information element that collects things like application name, version, and option flags.
- An internal communication element of the application that describes how the application does internal communication. Input, output, and error fields are described. These child fields contain subfields that can be used to describe the field and bindings to particular core services needed to read or write the file.
- An external communication element of the application that describes how the application communicates with its environment. This is not implemented, but is put as a placeholder.
- An execution bundle element that contains a list of core web services for the execution of the application. These services are stored by name, description (i.e. human comments for that service), definition (the URL for the WSDL file), and host bindings. The host bindings include a service binding point element that points to the actual SOAP server on the host that implements the service, and a Host description schema which is flexible field and is our Host XML Descriptor we used now

- An optional, generic parameter element to hold arbitrary information about the application that is not covered by the elements above.

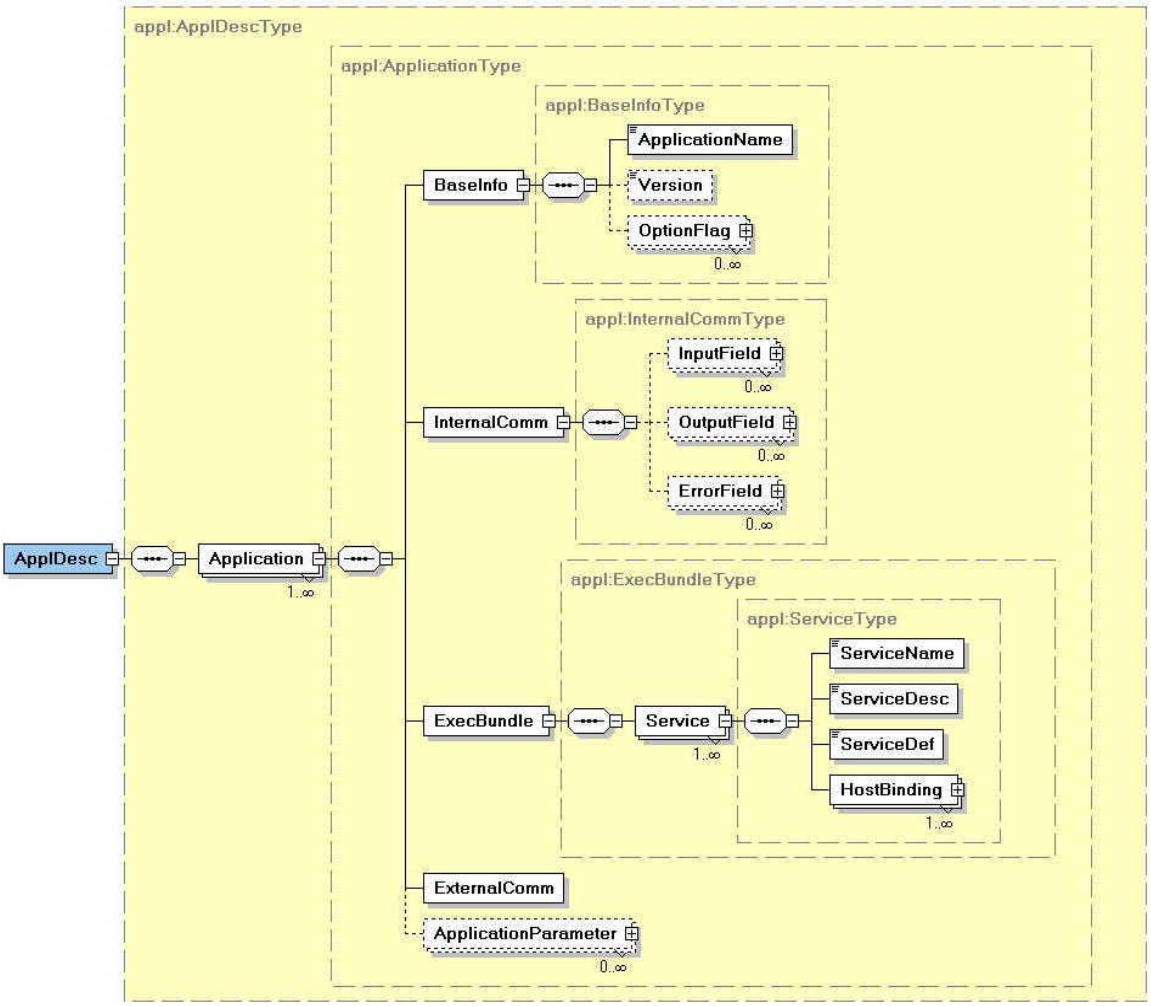


Figure 5.2: Schematic diagram for the abstract Application XML descriptor

In addition to the application schema, we define the host and queue binding description schema shown in Figure 5.3 which is modular container as our design mechanism. The host binding schema is available from B.4 and B.5 of the Appendix for more information in detail and contains the following elements about resources:

- The host information element such as DNS name and IP address.

- The application information element need to invoke the actual application code running on that host such as location of the executable code, location of the workspace or scratch directory.
- Queue information element such as the location of the executable queue command running on that host and the queue type which the host supports.
- Like the application schema, for maximizing the flexibility, we also provide a general purpose "parameter" element that allows for arbitrary name-value pairs to be included. This can be used for example to specify environment variable settings needed on a particular host by a particular application.
- This schema also has a queue binding element that contains information needed to perform queue submissions.

The queue binding schema is available from B.6 and B.7 of the Appendix for more information in detail and contains the queue script information (the job name, user account name, memory size, the number of CPUs, the wall time, the email options and the general purpose parameter for the queue) based on the queuing system, such as PBS. These schemas are designed to be plugged into the application descriptor schemas, and may be replaced by other schema definitions.

Similarly, the application instance XML descriptor is generated from the abstract application XML descriptor by the user's interaction in the portal. This schema is available from the B.3 of the Appendix and basically, followed by the application schema which describes only one application, one host binding, and one queue binding on the host.

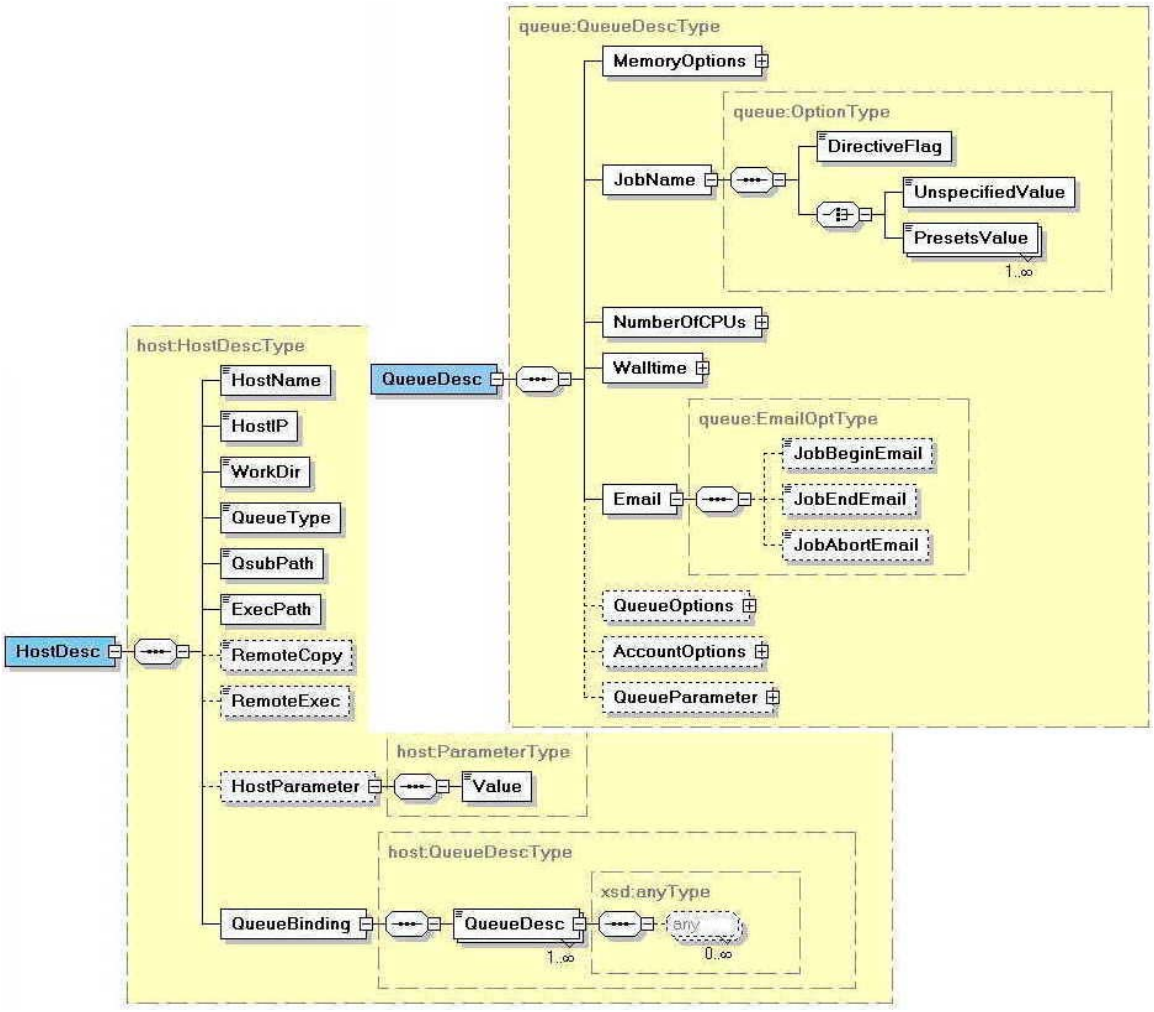


Figure 5.3: Schematic diagram for the abstract Host and Queue XML descriptors

5.5 The AWS deployment and use

The application metadata must now be turned into a useful service. The schemas themselves can be mapped to Java language bindings automatically using tools such as Castor [65]. The schemas are too complicated, however, to be used to define a useful WSDL interface, so we instead implement “Façade” wrapper classes that simplify access to the schemas, at the loss of some functionality. The wrapper interfaces are still quite large, and the WSDL for the wrapper interface definition is given in the A.6 of the

Appendix for the application abstract descriptor and the A.7 of the Appendix for the application instance descriptor.

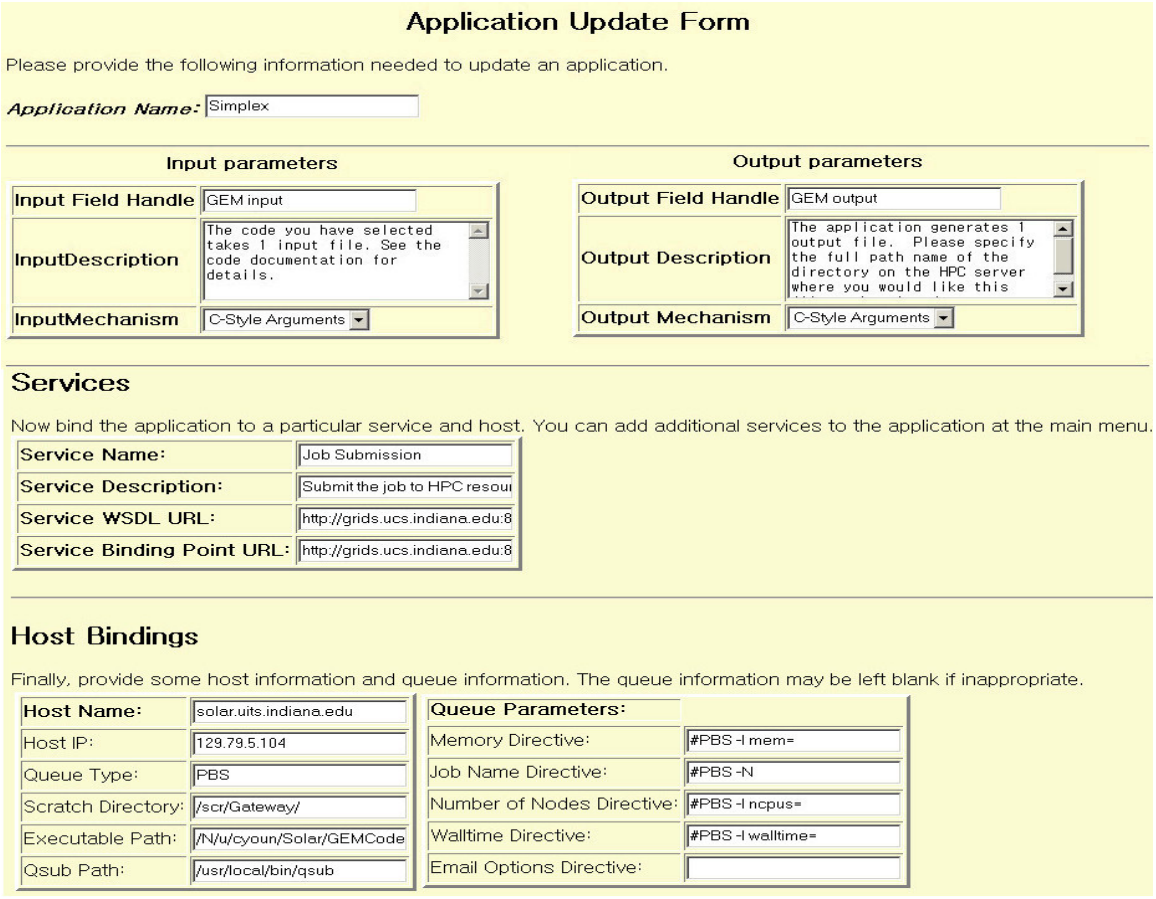


Figure 5.4: Sample application administration view for the application, Simplex code

We now review how to deploy an application as a Web service. Clients and user interfaces may be built out of these interfaces. Figure 5.4 illustrates part of an interface for the application manager. Application managers may provide various pieces of information needed to create instances of Abstract Application Descriptors. The illustrated form indicates the simple case of adding an application with one input and one output field, no command line flags, etc. The application can then be bound to a set of services on various hosts for submission and file transfer. The form elements are mapped to client stubs, which in turn (via SOAP) can be used to view and modify remote AWS

schema instances. The generated abstract application XML descriptor is shown in Table 5.1, 5.2, and 5.3. For managing applications, we provide three capabilities such as adding, updating, and deleting page.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:ApplDesc xmlns:ns1="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Appl2">
 <ns1:Application>
  <ns1:BaseInfo>
   <ns1:ApplicationName>Simplex</ns1:ApplicationName>
   <ns1:Version>Ver_0.0.1</ns1:Version>
  </ns1:BaseInfo>
  <ns1:InternalComm>
   <ns1:InputField>
    <ns1:Handle>GEM input</ns1:Handle>
    <ns1:Description>The code you have selected takes 1 input file. See the code documentation for
details.</ns1:Description>
    <ns1:IOMechanism>
     <ns1:localMech>CArgument</ns1:localMech>
    </ns1:IOMechanism>
   </ns1:InputField>
   <ns1:OutputField>
    <ns1:Handle>GEM output</ns1:Handle>
    <ns1:Description>The application generates 1 output file. Please specify the full path name of the
directory on the HPC server where you would like this file to be placed.</ns1:Description>
    <ns1:IOMechanism>
     <ns1:localMech>CArgument</ns1:localMech>
    </ns1:IOMechanism>
   </ns1:OutputField>
  </ns1:InternalComm>
  <ns1:ExecBundle>
   <ns1:Service>
    <ns1:ServiceName>jobsubmit</ns1:ServiceName>
    <ns1:ServiceDesc>Submit the job to HPC resources directly.</ns1:ServiceDesc>
    <ns1:ServiceDef>http://grids.ucs.indiana.edu:8045/GCWS/
                services/Submitjob?wsdl</ns1:ServiceDef>
    <ns1:HostBinding HostName="solar.uits.indiana.edu">
     <ns1:ServiceBindingPoint>http://grids.ucs.indiana.edu:8045/GCWS/
                        services/Submitjob</ns1:ServiceBindingPoint>
       <!-- Host Description biding area which is pluggable -->
    </ns1:HostBinding>
   </ns1:Service>
  </ns1:ExecBundle>
 </ns1:Application>
</ns1:ApplDesc>
```

Table 5.1: The abstract application descriptor for the Simplex application code

We described above how to deploy an application web service but not how to use it. The application XML descriptor gives a user various choices and are used to generate

forms needed to collect the metadata about a particular application. By filling this web form out, the particular invocations of an application, that is, an Application instance descriptor, are created in a separate set of schema which is different from the application XML descriptor. An application instance XML descriptor can be used for making the job script through the whole AWS lifecycle except the abstract application stage and form the backbone of a session archiving system, which allows users to view and edit old sessions.

```
<ns1:HostDesc xsi:type="HostDesc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ns2:HostName xmlns:ns2="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    solar.uits.indiana.edu</ns2:HostName>
  <ns3:HostIP xmlns:ns3="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    129.79.5.104</ns3:HostIP>
  <ns4:WorkDir xmlns:ns4="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    /scr/Gateway/</ns4:WorkDir>
  <ns5:QueueType xmlns:ns5="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    PBS</ns5:QueueType>
  <ns6:QsubPath xmlns:ns6="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    /usr/local/bin/qsub</ns6:QsubPath>
  <ns7:ExecPath xmlns:ns7="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
                    /N/u/cyoun/Solar/GEMCodes/Simplex/simplex</ns7:ExecPath>
  <ns8:QueueBinding xmlns:ns8="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
     <!-- Queue Description biding area which is pluggable -->
  </ns8:QueueBinding>
</ns1:HostDesc>
```

Table 5.2: The host descriptor for resource specification of the Simplex application code

As stated at Chapter 7, this form shown in Figure 7.9 is generated from the Application XML descriptor for a particular application runs: the input files used, the location of the output, the resources used for the computation, etc. At this point, the state transition from the application description to the application instance is occurred. Thus, an Application Instance XML descriptor is created and is deployed as a service.

Application metadata is intended to serve as a supplemental, specialized data model that is more appropriate for Application Portals than WSIL or UDDI. We believe this has an important role in computational services. The Open Grid Services Infrastructure

(OGSI) specification provides an important mechanism for describing particular service metadata [63]. On top of this we need service classification systems, which organize WSDL definitions in meaningful ways. Such classification schemes are subjective and not likely to be fully standardized, so we anticipate a future interesting problem of managing many different application metadata ontologies.

```
<ns8:QueueDesc xsi:type="QueueDesc">
  <ns9:MemoryOptions xmlns:ns9="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns9:DirectiveFlag>#PBS -l mem=</ns9:DirectiveFlag>
    <ns9:UnspecifiedValue>64mb</ns9:UnspecifiedValue>
  </ns9:MemoryOptions>
  <ns10:JobName xmlns:ns10="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns10:DirectiveFlag>#PBS -N </ns10:DirectiveFlag>
    <ns10:UnspecifiedValue>simplex</ns10:UnspecifiedValue>
  </ns10:JobName>
  <ns11:NumberOfCPUs xmlns:ns11="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns11:DirectiveFlag>#PBS -l ncpus=</ns11:DirectiveFlag>
    <ns11:UnspecifiedValue>1</ns11:UnspecifiedValue>
  </ns11:NumberOfCPUs>
  <ns12:Walltime xmlns:ns12="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns12:DirectiveFlag>#PBS -l walltime=</ns12:DirectiveFlag>
    <ns12:UnspecifiedValue>00:00:11</ns12:UnspecifiedValue>
  </ns12:Walltime>
  <ns13:Email xmlns:ns13="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns13:JobBeginEmail/>
    <ns13:JobEndEmail/>
    <ns13:JobAbortEmail/>
  </ns13:EmaiI>
  <ns14:QueueParameter Name="#PBS -M "
xmlns:ns14="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <ns14:Value>gateway@grids.ucs.indiana.edu</ns14:Value>
  </ns14:QueueParameter>
</ns8:QueueDesc>
```

Table 5.3: The queue descriptor for PBS resource of the Simplex application code

Chapter 6

Web service Security and Negotiation

6.1 Introduction

Scientists and researchers can interact with the computational Grids such as Globus via portal mechanisms [11]. For accessing those remote resources, one can view the Web Services approach [22,23,72] as a "kit" of services, which can be used in different ways in different applications, introduced in Chapter 4. Seamless access in a computing portal is an important service, which is aimed at allowing applications to be run on arbitrary appropriate backend resources. Such seamless capability is rather natural in a computing web-based architecture.

Basically, a computing portal [73] is an application that integrates access to the data, computers and tools needed for a particular computational science area. It comprises an infrastructure enabling scientists to use a diverse set of distributed services that access

remote compute resources. We may identify the general services needed for the computing portal. First, a security service allows the authenticated user to use the distributed remote resources through the portal. Second, a discovery service such as object lookup and registration allows a user to find the usable and available service in the portal. Third, an information service such as object persistence and database support allows a user to store and recover the portal data. Fourth, a job tracking service such as event and transaction allows a user to monitor and track the job status from the batch scheduler. Fifth, a file service provides a user for managing the user directory such as copy, rename, move, and so on. Sixth, a job composition service such as application integration allows a user to compose the core services. Finally, there are collaboration services [74].

Based on these service capabilities in the portal, we described how we design and implement core Web services and Application Web services in Chapter 4, 5 which is focus on the service-oriented architecture for processing the complex scientific problem solving. As one example, using the Application Web Services which is described in Chapter 5, legacy applications can be easily wrapped into the portal services that are available to appropriate users, including other portals. From the user's point of view, it is very useful for the user to adapt the scientific applications into the portal.

Security in such an environment is critical, since it often involves directly accessing a user's resources on a particular computer through delegation to a middle tier proxy. In this chapter, we describe how message level security may be incorporated into the architecture of a Web service based computing portal. We must address two important problems. First, how do we add security to Web service messages, over and above

transport level security? Second, how can we provide an adaptable system that works with different security message formats and mechanisms? We need secure SOAP messages between user interface server and the repository and the service provider for the user authentication, based on the message-level security architecture. SOAP security should be provided through standard interfaces to specific mechanisms such as Kerberos [57], Shibboleth [75], PKI [56], or Globus GSI [58]. The general approach is to use the assertion based security such as SAML, WS-security into SOAP messages. An assertion, for example, SAML, WS-Security, is an XML document describing the information about authentication acts performed by subjects, attributes of subjects and authorization decisions, created with a specific mechanism. To this, we must add additional metadata that can be used to forward the secure assertion to the correct handler classes.

Next, managing multiple versions of services is an important consequence of this issue, and we close with a description of our work on negotiation for version control. As one example, version negotiation is a prototypical approach for solving the problems of service compatibility.

6.2 Secure Web services

Computational Web portals may be built out of Web services such as job submission, context management, file service and application Web services which are described in the Chapter 4, 5. Access to remote resources is performed through portal services that consist of Web services acting as the middleware. These Web services pieces make no provisions for authentication, message integrity, authorization service (access control), and other security concerns.

GSI [58] is based on the public key infrastructure which is used for X.509 certificates, and the Secure Sockets Layer (SSL) communication protocol. For this, GSI SOAP implementation [76] is available, which provides a GSI enabled HTTP protocol, GSI delegation, and authentication capabilities.

SOAP-based Web services do not directly provide message-level security but do provide an extensibility mechanism: arbitrary additional XML messages such as security assertions may be included in SOAP headers. We are investigating a general purpose way of securing SOAP messages that support multiple underlying mechanisms such as Kerberos, PKI, and GSI independently based on user assertions using SAML, the Secure Assertion Markup Language [77].

We have developed a prototype system to support a secure Web services through the single sign-on (SSO). SSO allows the use of a secure resource at the destination site without directly authenticating to it, and requires secure delegation of initial authentication credentials. As the example of our prototype system, we have used Kerberos as a security mechanism and SAML as an assertion, but we have attempted to keep our design general for the adaptive control and will add support for other mechanism such as PKI and Globus GSI.

6.2.1 Web services security languages

SAML [77], an OASIS standard, is an XML-based security services framework for exchanging authentication and authorization information. Assertions are mechanism-independent, digitally signed information about authentication acts performed by subjects,

attributes of subjects, and authorization decisions about whether subjects are allowed to access certain resources.

```
<?xml version="1.0" encoding="UTF-8"?>
<ns1:AssertionSpecifier xmlns:ns1="http://www.oasis-open.org/committees/security/docs/draft-sstc-schema-assertion-27.xsd">
  <ns1:Assertion MajorVersion="1" MinorVersion="0" AssertionID="156.56.104.10.1054324468957"
    Issuer="Gateway Web Portal" IssueInstant="2003-05-30T14:54:28.957-05:00">
    <ns1:Conditions NotBefore="2003-05-30T14:49:28.957-05:00" NotOnOrAfter="2003-05-30T15:04:28.957-05:00">
      <ns1:AudienceRestrictionCondition>
        <ns1:Audience>http://www.gatewayportal.org/agreement.xml</ns1:Audience>
      </ns1:AudienceRestrictionCondition>
    </ns1:Conditions>
    <ns1:AuthenticationStatement
      AuthenticationMethod="urn:ietf:rfc:1510" AuthenticationInstant="2003-05-30T14:54:28.957-05:00">
      <ns1:Subject>
        <ns1:NameIdentifier
          SecurityDomain="www.gatewayportal.org" Name="gateway"/>
        <ns1:SubjectConfirmation>
          <ns1:ConfirmationMethod>urn:ietf:rfc:1510</ns1:ConfirmationMethod>
          <ns1:SubjectConfirmationData>A Kerberos Ticket</ns1:SubjectConfirmationData>
        </ns1:SubjectConfirmation>
      </ns1:Subject>
      <ns1:AuthenticationLocality IPAddress="156.56.104.10" DNSAddress="grids.ucs.indiana.edu"/>
    </ns1:AuthenticationStatement>
    <ns1:AuthorizationDecisionStatement Resource="AccessLevel" Decision="Permit">
      <ns1:Subject>
        <ns1:NameIdentifier
          SecurityDomain="www.gatewayportal.org" Name="gateway"/>
        <ns1:SubjectConfirmation>
          <ns1:ConfirmationMethod>urn:ietf:rfc:1510</ns1:ConfirmationMethod>
          <ns1:SubjectConfirmationData>A Kerberos Ticket</ns1:SubjectConfirmationData>
        </ns1:SubjectConfirmation>
      </ns1:Subject>
      <ns1:Actions>
        <ns1:Action>5</ns1:Action>
      </ns1:Actions>
    </ns1:AuthorizationDecisionStatement>
  </ns1:Assertion>
</ns1:AssertionSpecifier>
```

Table 6.1: SAML example for the Kerberos based security mechanism

WS-Security [78] is a Web services security language as the basis for the construction of a wide variety of security models including PKI, Kerberos, and SSL and encryption technologies. It enables applications to construct secure SOAP message exchanges and

describes a single-message security language that provides for message security that may assume an established session, security context and/or policy agreement.

Our authentication system is based on SAML as Web services security language. Although not included in the prototype, we have designed it to support other Web service security language such as WS-Security. A SAML-consuming service can accept the signed assertion or use it to find locations of Kerberos proxy tickets or grid proxy certificates. SAML can also be used to convey access control decisions made by other mechanisms, such as Akenti [79,80]. SAML assertions are added to SOAP messages. We have implemented the SAML specification schema in Java using Castor [65], which can be used to convert between XML schemas and Java data objects. We have also developed client and server applications for handling SOAP with SAML messages. Table 6.1 describes an example of SAML assertion generated by the assertion generator after the user authentication using the security mechanism, Kerberos.

6.2.2 Secure SOAP message format

We are implementing a message-level security system using the Web services security language such as SAML and WS-Security, and the mechanisms such as Kerberos, PKI, Globus GSI. In order to support these different formats and mechanisms, we have added additional metadata to the SOAP header: we configure the message format which can identify the assertion and security mechanism.

As the language identifier, we are using the name tag, “Saml” for SAML and “WSSecurity” for WS-Security. This is followed by a tag “SignedAssertion” that contains the signed and perhaps encrypted security assertion. According to this name tag,

SOAP server can extract and parse the user information and authorization for resources. In order to encrypt and decrypt the signed SOAP message (signed assertion and SOAP body message), we also need to specify what kind of the security mechanism is used to get a shared key session object. We thus provide an additional tag, “SecurityMechanism”, “Kerberos” for Kerberos, and “PKI” for PKI. The “UserName” tag is used for checking the user secure key object as the user identifier in section 6.2.4.

Table 6.2 describes the example of the message format which is based on SAML as the language identifier and Kerberos as the security mechanism.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Header>
  <ns1:Saml xmlns:ns1="http://www.gatewayportal.org/sign.xsd">
   <ns1:SignedAssertion>
         YIIIYwYJKoZIhvcSAQICAgEAAAAA//9a+0MDxeg14f8T5vf0o7jm9z4ml2Fj
          azlhwxyd/kZz8pgWbREMMZF2ELm9G+MFojzGKt0F6B91gBuJ1QL+QN5kM
                              ……..
                              ……..
          n8cdEhjskpcEYP2MvnRwxJmei9U5m3IToiHDI3foZ2TjhwPn
   </ns1:SignedAssertion>
   <ns1:SecurityMechanism>Kerberos</ns1:SecurityMechanism>
   <ns1:UserName>gateway@CG.INDIANA.EDU</ns1:UserName>
  </ns1:Saml>
 </soapenv:Header>
 <soapenv:Body>
  <ns2:SignedBody xmlns:ns2="http://www.gatewayportal.org/signbody.xsd">
        YIIBawYJKoZIhvcSAQICAgEAAAAA//9NPq5TRhFcyfMdAYFS1XHlBzI3JhN+c15
        z0MncshbXb9zQ3Z8b6QkJHCmWZuVBXvyVSaDZ4GVgbwnicAJSGEj6OJVTxqQfH
                    ……
                    ……
        YFKg/LQJ0oleULSsP2k9HQY+MxD64IYaw9lSVOX9IHtc+uZyQ==
  </ns2:SignedBody>
 </soapenv:Body>
</soapenv:Envelope>
```

Table 6.2: Message format for secure Web services

6.2.3 Message-level security infrastructure

The previous sections addressed the problem of adapting to multiple mechanisms and security formats. Next, we must describe mechanisms for actually creating the secure messages and assertions. Our prototype authentication system for using Kerberos and SAML is illustrated in Figure 6.1 and works as follows:

In client-side process, we developed utility classes using SAML java objects for creating assertions. Specific assertions are created using the appropriate security mechanism. SAML assertions are marshaled back and forth between Java and SAML XML representations. The procedures for generating a secure client message are as follows:

(1) A user logs in through a web browser and gets the authentication and a Kerberos ticket on the User Interface Server (UIS).

(2) As the user's request, UIS chooses the security mechanism and security markup language. It then establishes the security context with the server for getting the shared key. For example, the Kerberos client on the UIS creates a client session object that contacts a Kerberos server in a session object. The client and server then establish a GSS [81] context, which is maintained on each server in user sessions. Each of these objects possesses one half of the symmetric key pair for a particular user.

(3) The Assertion Generator on UIS generates a particular user assertion, for example, user's SAML security assertion.

(4) The Signature Generator on UIS signs the user assertion and message (a SAML assertion in SOAP header and SOAP Body messages).

(5) UIS rebuilds the SOAP message which is described by Section 6.2.2 to include

the additional security information in the header.

(6) Finally, the SOAP request is sent to the desired SOAP server which provides computing services.

In server-side process, SOAP messages are extracted and parsed into several parts such as user's assertion, security mechanism, security markup language, and SOAP body message.

(1) The SOAP server establishes a security context with the client for getting the shared session key that will be used for unwrapping the secure messages. It does not check the signature of the request directly and instead forwards to the authentication service shown in Figure 6.1, which verifies the signature. For example, the Kerberos server responds positively or negatively to the SOAP server, which may then fulfill the client's request.

(2) The SOAP server handles the incoming SOAP request message. It extracts the secure assertion message for the SOAP header, the secure body message from the SOAP body, and the security mechanism name such as Kerberos, PKI, and the security message format such as SAML, WS-security.

(3) The SOAP server checks the message format and the security mechanism. It then uses the Unwrap Generator class, which includes the server key object on SOAP server, decrypts a signed SAML assertion and SOAP body message.

(4) The validity of the message is checked. For example, SAML fields such as issuer name, "conditions" time limit, subject name, and authorization may all be verified.

(5) If the message is determined to be valid, the SOAP server reconstructs the

original SOAP messages (removing the security header) and passes this to the server's SOAP engine for processing.

This system illustrates the basic client-server interaction and can be used to build a single sign-on system. As one example, using this infrastructure, we have developed the specialized use for SAML and Kerberos. We are currently also considering PKI based authentication system.

As the technical resources for our implementation, we have used the SOAP engine, Apache Axis 1.0 [39] which is modified for adding the security process, SAML schema (draft-sstc-schema-assertion-27.xsd) [77], Kerberos, Version 5, release 1.2.2, and Java 1.4 or higher.

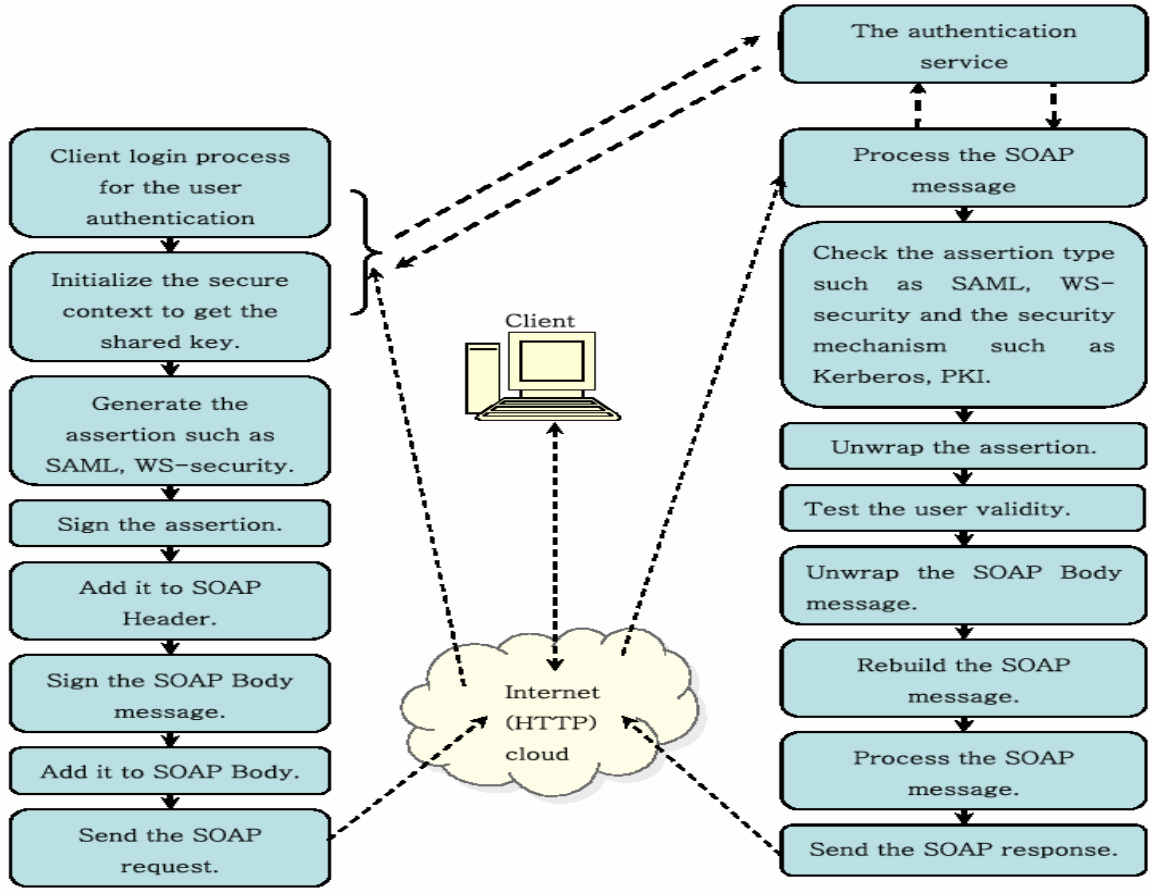


Figure 6.1: An assertion-based security infrastructure

6.2.4 Multiple accesses in a distributed system

Usually, User Interface Server in computing portals federates a bunch of Web service proxies for accessing distributed services. When we use the client-server interaction fashion shown in Figure 6.1, the client has each secure session object on each distributed service. In this case, we need more effective system for handling client's secure session object.

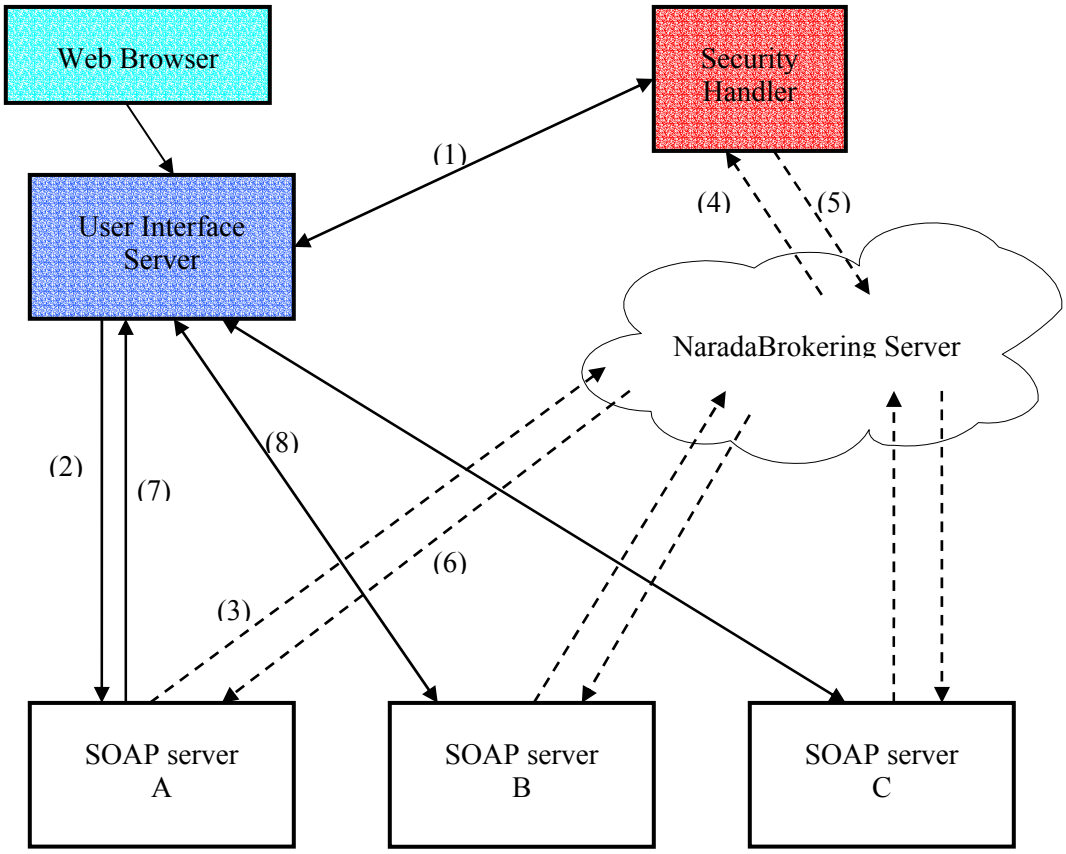


Figure 6.2: Interactions of secure Web service in a distributed environment.

Figure 6.2 illustrate multiple accesses in a distributed system, as separating a secure server session object from an SOAP server which is running a bunch of Web services. It is possible to use the messaging or event system, so called the Narada event brokering

system developed by Community-Grids Lab. at Indiana University [82]. Narada is a distributed event brokering system designed to run on a large network of cooperating broker nodes. Communication within Narada is asynchronous and the system can be used to support different interactions by encapsulating them in a specialized events. NaradaBrokering also provides JMS compliance which follows the well-known publish-subscribe model.

Using NaradaBrokering system which is a messaging middleware, clients can interact with distributed computing services securely. In Step 1 as indicated in Figure 6.2, a client logs in through a web browser and authenticates with the specified security mechanism and results in getting a Kerberos ticket or the user certificate on the User Interface Server (UIS). It then establishes the security context with the "Security Handler" subscriber for getting the shared key. As described in Figure 5.1, UIS makes secure SOAP message and then invoke the desired one of distributed services in Step 2. In Step 3, the selected SOAP server (SOAP server A) for using the Web service extracts SOAP Header message and SOAP Body message, respectively from the secure SOAP message and then publishes them into the NaradaBrokering server. Those messages are processed in Step 4. "Security Handler" subscriber establishes and maintains a security context with the client for getting the shared session key that will be used for unwrapping the secure messages which is coming from distributed SOAP servers. It also checks the validity of the user assertion. Following with the decision-making process, we currently provide two failures decision levels such as "User assertion is NOT valid" and "User's secure key object is NOT generated". In Step 5, "Security Handler" subscriber publishes the SOAP message with the decrypted SOAP Body message which is rebuilt in into the NaradaBrokering

server if the test results for this user are valid. The selected SOAP server takes the SOAP message through the NaradaBrokering server and then makes a process it in Step 6. Finally, UIS get the SOAP response message as the user's service request in Step 7 and then try to interact with another distributed service without authenticating again in Step 8, following with the above procedures.

6.3 Web service negotiation

Distributed service systems such as we are building will need to address many issues as they move from demonstrable prototypes to production systems. One crucial problem is negotiation of quality of service. Next, service compatibility is a related problem: assuming the model depicted in Chapter 4, the client must first determine if it has the appropriate stubs to invoke the discovered service. Alternatively, the client and server may support various versions of a particular service and will want to negotiate the optimal version between them.

6.3.1 Overview

Resource management needs good network environments, regardless of available bandwidth. A network needs to be managed well for using computing resources more effectively. Similarly, a network needs solid management to make sure that required resources are given for most critical applications. In a network environment, Quality of Service (QoS) is an essential element of the well-managed network, bringing resource predictability and availability as a service to applications, and the set of techniques designed to manage bandwidth, delay, jitter, and packet loss in a network.

Like this, Quality of service negotiation for Web service as an important facet of Web services' functioning needs to take place between all interacting parts. A classic example here is file transfer service that has different bindings for default usage, high performance, and reliability, respectively. The actual service used would need to be negotiated between the parties. Web services don't currently provide this notion, but examples of negotiation protocols abound, from SIP and H.323 in AV to SSL handshakes in security.

Another taking advantage of this is service compatibility. Thrift has required that new Web services be backward compatible. That is, new Web services must be able to work with existing Web services. This is important because Web services for computing portal are in a constant state of evolution. Piecemeal upgrades are the rule and necessary. For example, due to the pace of change in the industry for the computer software today, people are not keeping up with the latest developments and running out to buy the newest version of computer software as soon as it comes out. They don't want to upgrade and replace the old version of the computer software as new one, even though it is very good working. It means that lots of people are backward in the software they are using, without updating their software in years. So, no matter how backward you are, you are still able to use new stuff without worrying about whether it is compatible or not.

Web service negotiation enables a user to provide better service to certain selected Web services, for instance, job submission, file transfer manipulation and the application Web services. We describe the general approach of it as Quality of Service (QOS) negotiation and service compatibility for computing services, adapting computing environments. Service providers have different requirements for Web service negotiation and it is interpreted in different ways. For example, Web service negotiation to service

providers means the ability to offer different negotiations of the service so that they can serve their clients differently. It can also mean the version control ability to transport the different version of computing Web Service over the same computing infrastructure.

We currently describe some existing technologies which are used for the negotiating mechanism, including SSL handshake, SIP, and intelligent software agents. As a model for negotiating service, offer/answer approach is introduced. Finally, we present the design and implementation issues for Web service negotiation.

6.3.2 Motivating examples

We briefly review here some motivating examples that we considered when designing a prototype system for Web services.

The *Secure Socket Layer (SSL) handshake protocol* [59,83] is used to negotiate the cipher suite first and then begins with an exchange of information between the client and server. That is, the goal of the SSL Handshake between the client and the server is to negotiate on an acceptable protocol version such as v2 or v3, and to select the appropriate set of cryptographic algorithms, i.e., cipher and hash methods, and to authenticate uni- or bi-directionally using PKI certificate, and to securely distribute shared secrets for exchanging the data. The SSL handshake happens every time a client starts a session with a "new" server or when either party for whatever reason, regardless of any reason, wishes to establish a new SessionID and Cipher Specification.

The *Session Initiation Protocol (SIP)* [84,85] is a standard of Internet Engineering Task Force, especially for Voice over IP and an application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. These multimedia

sessions include multimedia conferences, distance learning, Internet telephony and similar applications. There is a need to negotiate a multitude of parameters, settings, and algorithms for example, compression algorithms, encryption algorithms, code book size, and message integrity mechanisms when setting up sessions using SIP. This negotiation would take place prior to session establishment, between any two SIP entities. The result of the negotiation is a key that will be used in subsequent transactions to maintain the negotiation state. This key is carried as a SIP header ("Key" field) in further SIP messages.

As of intelligent software agents, the Semantic Web [86] is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation and represent data based on eXtensible Markup Language (XML) and the Resource Description Framework (RDF). XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean. Meaning is expressed by RDF [87], which is to make it possible to specify semantics for data based on XML in a standardized, interoperable manner and provides interoperability between applications that exchange machine-understandable information on the Web.

Software agents which support more machine-readable Web content and automated services (including other agents) using the Semantic Web are more effective and intelligent when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. Agents between client and service provider can reach a shared understanding by exchanging ontologies, which is a document or file that formally defines the relations among terms.

Agents can even "bootstrap" new reasoning capabilities when they discover new ontologies. Semantics also makes it easier to take advantage of a service that only partially matches a request. For example, suppose service provider has the different versions of the computing service. If a client sends a request, “give me 1.0 or higher”, or “give me 2.0 or higher but not 2.1”, to the server those expressions are more human-readable. They are easier way to make a negotiation between a web client and service provider using intelligent software agents

6.3.3 Offer/Answer model

Web service was basically conceived as a way to describe RPC-based XML messaging system which is more accessible and readable to human. SOAP was designed as a unicast mechanism to carry SOAP messages. As a result, even if SOAP has expressiveness to describe a distributed messaging system, it is missing information from both participants, and agreement on parameters between them.

As a remedy, Web service negotiation follows an offer/answer approach [88]. These two protocols such as like SSL handshake and SIP negotiate method are dissimilar in application but alike in their implementation of negotiation: both use an offer/answer approach. This model may be readily extended to Web services. We suggest that SOAP over HTTP is the appropriate protocol for the negotiation phase, regardless of the actual protocol used for actual service invocation. In this model, one participant in the session generates a SOAP message that constitutes the offer. For example, an offer might include a set of desired protocols and interface versions that the offerer wishes to use. The offer is conveyed to the other participant, called the answerer. The answerer responds with a

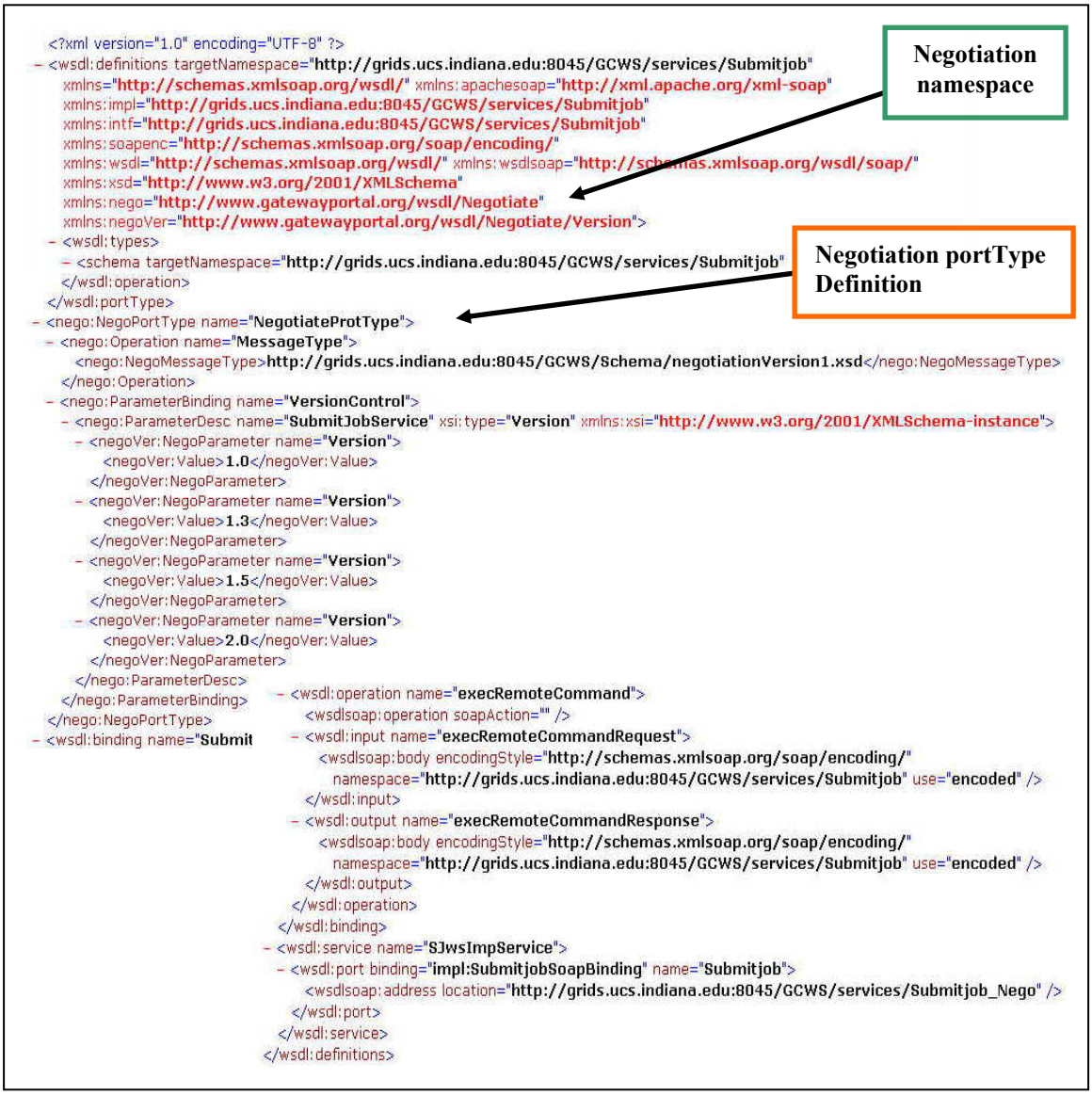
SOAP message to the offerer. The answer indicates whether the service request is accepted or not, and if accepted, the selected parameters from the offer. This offer/answer model is most useful in Web service negotiation where information from both participants is needed for negotiating some parameters.

6.3.4 Implementation

We may implement Web service negotiation through extensions to WSDL. In particular, a given WSDL interface can contain multiple portTypes. For example, multiple versions of the same service may be specified in separate portTypes in the same WSDL document. This example is shown in the Table 6.3. In our prototype system, WSDL descriptions that support negotiation between various portTypes must define one additional, specialized portType for negotiation. Before using a particular Web service such as job submission, the WSDL PortTypes that describes the service should be taken with the WSDL negotiation PortType. Web service negotiation is an additional "PortType" and Operation that provide interfaces for creating negotiation service.

We implement Web service negotiation as a supplemental portType (in a separate name space) that is included in the WSDL service description as an extension. Given the wide range of possible uses, we have attempted to make this modular, which would allow "standard" negotiation descriptions to be plugged into the portType frame work. The schema for this portType shown in Figure 6.3 and the description can be obtained from the B.8 of the Appendix in detail. The negotiation portType contains two elements: "operation" and "parameters". The "operation" element is intended to be extended by

another URI that defines a standard negotiation message format. The "parameters" element is extended to contain the actual data used in the negotiation.




```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
    xmlns:impl="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob"
    xmlns:intf="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:nego="http://www.gatewayportal.org/wsdl/Negotiate"
    xmlns:negoVer="http://www.gatewayportal.org/wsdl/Negotiate/Version">
  - <wsdl:types>
    - <schema targetNamespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob"
    </wsdl:operation>
  </wsdl:portType>
- <nego:NegoPortType name="NegotiateProtType">
  - <nego:Operation name="MessageType">
      <nego:NegoMessageType>http://grids.ucs.indiana.edu:8045/GCWS/Schema/negotiationVersion1.xsd</nego:NegoMessageType>
    </nego:Operation>
  - <nego:ParameterBinding name="VersionControl">
    - <nego:ParameterDesc name="SubmitJobService" xsi:type="Version" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      - <negoVer:NegoParameter name="Version">
          <negoVer:Value>1.0</negoVer:Value>
        </negoVer:NegoParameter>
      - <negoVer:NegoParameter name="Version">
          <negoVer:Value>1.3</negoVer:Value>
        </negoVer:NegoParameter>
      - <negoVer:NegoParameter name="Version">
          <negoVer:Value>1.5</negoVer:Value>
        </negoVer:NegoParameter>
      - <negoVer:NegoParameter name="Version">
          <negoVer:Value>2.0</negoVer:Value>
        </negoVer:NegoParameter>
      </nego:ParameterDesc>
    </nego:ParameterBinding>
  </nego:NegoPortType>
- <wsdl:binding name="Submit
    - <wsdl:operation name="execRemoteCommand">
        <wsdlsoap:operation soapAction="" />
      - <wsdl:input name="execRemoteCommandRequest">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob" use="encoded" />
        </wsdl:input>
      - <wsdl:output name="execRemoteCommandResponse">
          <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob" use="encoded" />
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="SJwsImpService">
    - <wsdl:port binding="impl:SubmitjobSoapBinding" name="Submitjob">
        <wsdlsoap:address location="http://grids.ucs.indiana.edu:8045/GCWS/services/Submitjob_Nego" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Table 6.3: An example WSDL for Web service negotiation

We suggest the approach is for all Negotiation operation schemas to have a hierarchical URI beginning with the name space similar to http://.../Negotiate. For example, the schema defining the standard format for file transfer protocol negotiation may be named http://.../Negoatiate/.../FTProtocol. A similar naming system can be

proposed for other uses, such as version negotiation for a particular service. The schema here defines the format for describing protocols. The actual protocols available for negotiation from a particular service are contained in the extension to the "parameters" element.

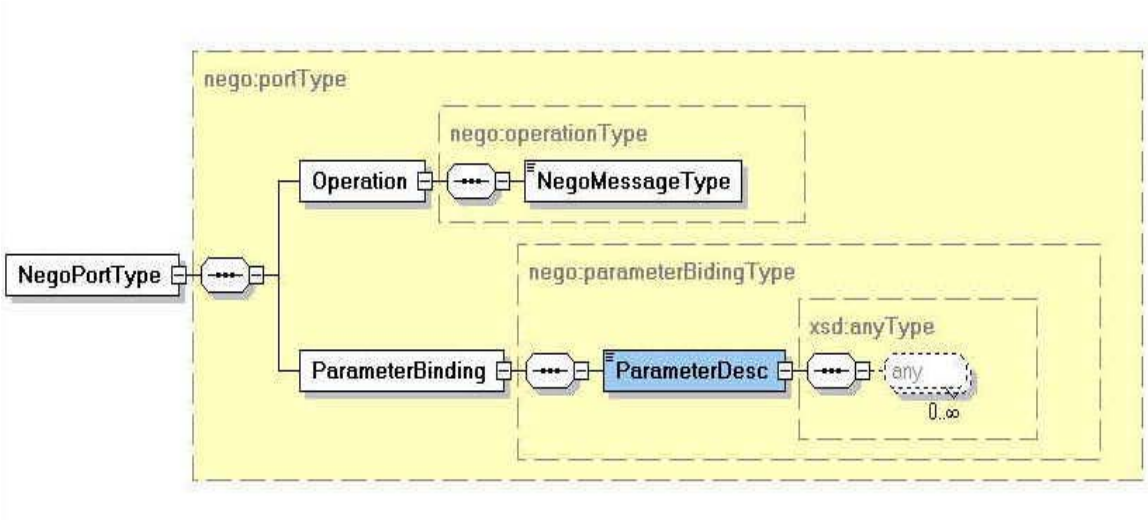


Figure 6.3: Schematic diagram for Web service negotiation

As a testing example for negotiation Web service, we describe family of parameter schemas for "standard" types of negotiation which is applicable to the negotiation descriptor: Version picking (namespace: http://.../Negotiate/../Version) shown in the B.10 of the Appendix and Protocol picking (namespace: http://.../Negotiate/../Protocol) shown in the B.9 of the Appendix. Each of these has a schema that extends basic parameter schema provided by the negotiation descriptor. Version picking schema shown in the B.10 of the Appendix contains a Version service name and a set of Version values.

The basic interaction of a Web service client with services for version control is as follows. For managing the operation messages between participants, we should pick a parameter family defined by a URI such as http://.../Negotiate/Version. A client sends its parameters configuration and the URI of that configuration to the service provider. The

service provider makes a decision when receiving the client's parameter list and URI for the negotiation information. The service provider selects the particular version based on any desired choosing algorithm. The service provider then sends the chosen version back to the client.

6.4 Summary

In this chapter, we have presented the design and implementation of secure computing Web services and the negotiation for Web services needed for the computing portal. Based on this message-level security system and Web service negotiation to determine quality of service and service compatibility, the portal developer can construct securely specific implementations and composites of primitive service components.

Following our initiation of this project, the National Science Foundation awarded a National Middleware Initiative grant to establish a reference implementation of SAML in Java and C++. This open source reference implementation is available from [89]. We intend to reconcile our implementation with this source.

Web service security is one aspect of our overall program for building Grid Computing Environments.

Chapter 7

Application: Distributed Earthquake Modeling Web Portal

In this chapter we demonstrate that we have built our earthquake modeling Web portal (QuakeSim) for the use by the seismological, crustal deformation, and tectonics communities for developing an understanding of active tectonic and earthquake processes using our Web service based computing portal architecture, introduced in Chapter 4, 5 and 6. The top-level operational architecture of our proposed solid earth research virtual observatory (SERVO), which makes a use of grid technologies, shows science users interacting with interface programs as well as modeling, simulation, and analysis tools. The goal of this project is to provide codes and data in an integrated Web services based environment using proxy component architecture. During that research and development project, explaining several science codes for earthquake modeling in detail is outside the scope of this dissertation and goes beyond our expertise. Our work presented here relies

on rapid prototyping, testing and deployment and concentrates on issues relevant to Web service based implementation.

7.1 Code and project descriptions

We describe several earthquake codes which have been implemented for individual use, or for interaction with other codes in QuakeSim portal system. A number of simulation methods for studying earthquakes are being developed by GEM consortium. The brief general code descriptions are below. A diagram indicating linkages between codes is shown in Figure 7.1.

1) **Disloc**: Handles multiple arbitrarily dipping dislocations (faults) in an elastic half-space to produce surface displacements.
2) **Simplex**: Inverts surface geodetic displacements for fault parameters using simulated annealing downhill residual minimization. It is based on disloc and uses dislocations in an elastic half space.
3) **Geofest**: Is a three-dimensional viscoelastic finite element model for calculating nodal displacements and tractions. It allows for realistic fault geometry and characteristics, material properties, and body forces.
4) **VC (Virtual California)**: Simulate interactions between vertical strike-slip faults using an elastic layer over a viscoelastic half-space.
5) **Park**: Is a boundary element program to calculate fault slip velocity history based on fault frictional properties.

6) **DAHMM**: Is a time series analysis program based on Hidden Markov Modeling to produce feature vectors and probabilities for transitioning from one class to another.
7) **PDPC**: Is a time series analysis pattern recognition program to calculate anomalous seismicity patterns and probabilities.

A complete code list and detailed descriptions may be found at [90]. As codes above become more robust and accepted, problems emerge as follows:

- We need to manage information about distributed data sources: multiple databases, sensors, simulated data.
- We need to organize, manage information about multiple code installation sites.
- We need to simplify access to data, use of codes, and use of visualization/analysis tools for broad range of users.
- We need to link application codes together.

The QuakeSim web portal, which is built out of portlet containers that access distributed resources via Web services, as described in Chapter 4 and 5, provides the unifying hosting environment for managing the various applications listed above. Each of the applications in Figure 7.1 are wrapped by an Application Web service proxy that can provide simple interactions with the hosting environments of the codes, allowing the codes to be executed, submitted to batch queuing systems, and monitored by users through the browser. Including job submission, file transfer and management, which are needed to run the application codes on some host in isolation, we must also support communications and interoperability for the input and output format between the codes listed above. The following motivating scenario [91] illustrates the interaction between

Disloc and Simplex code simulations, for example. A single InSAR satellite yields information on a one dimensional map of surface deformation which can be inverted to determine fault parameters consisting of location, depth, dip angle, strike angle, length and width, strike slip, dip slip, and tensile. Simplex code may take this surface displacement data and fits an initial guess of fault parameters that produce the surface deformation. One may test the potential value of additional InSAR data that would be to the fault modeling code. In particular, one may compare the improvement on errors in the fault models when one or two additional "look" angles are added. One additional "look" angle may be obtained from the satellite for collecting data in both the ascending and descending paths of the satellite orbit. However, two additional "look" angles would require the employment of an additional satellite. Instead of it, Disloc code may be used to generate the necessary synthetic InSAR data to simulate two and three additional data streams. As the simulation result, the additional synthetic data streams can improve the robustness of the fault parameter estimations. However, the difference between two and three "look" angles results in much less noticeable improvement.

Scenarios such as the above require some familiarity with the code in order to manage the code linkage. When we encounter the fully interacting system, displayed in Figure 7.1, we face the additional problem of scaling. We obviously want to support a more scaleable system with common data formats that may be shared between application codes. So, linking any two codes may be done in a one-time fashion through the common data format provided by the data Web service for each application codes. Code input/outputs may be translated in two stages, that is, from code A into the common format then into code B, and vice-versa. Also, from the portal architecture point of view,

linking multiple codes becomes possible to develop general purpose tools for manipulating the common data elements, defining XML schema and a well-defined framework for adding applications that is able to share data.

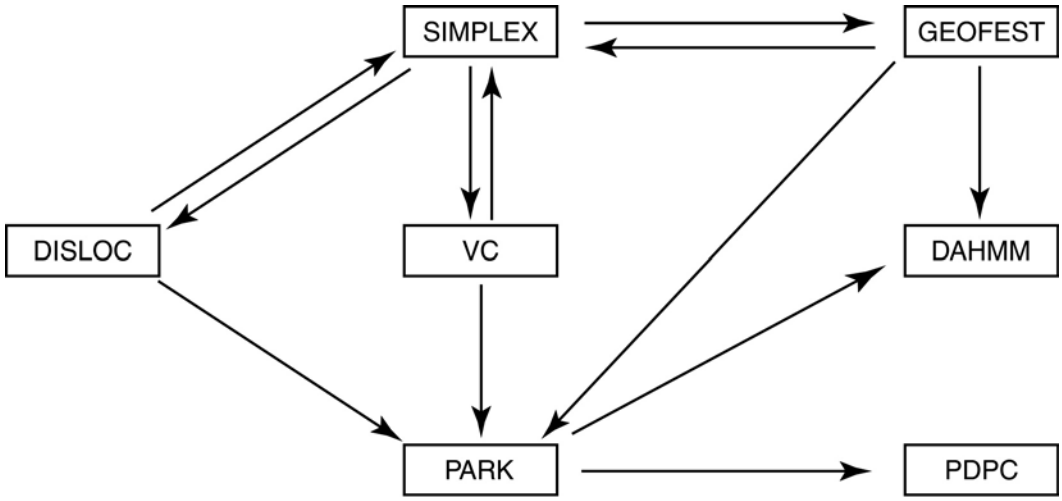


Figure 7.1: Linkages between codes that maybe run separately or coupled to other codes

7.2 QuakeSim Portal Architecture

GEM applications require a set of core computing services and Application Web service in QuakeSim portal, illustrated in Figure 7.2. Consequently, we need to deploy host-specific services such as Job submission, Job monitoring, File Transfer, and Data manipulation on each machine, Host 1, 2, and 3 with the web server for submitting and monitoring jobs, running applications, handling files, or formatting legacy input/outputs. And we also need to deploy host-independent services such as Context Manager, Script Generator, and Application Web service on a particular host with the web server for handling the metadata.

The user interacts with the portal through a web browser, which accesses a central user interface server. This server maintains several client API calls for accessing various

remote services. These remote services are described in the Web Service Description Language (WSDL) and are invoked via Simple Object Access Protocol (SOAP) invocations over HTTP. These services are invoked on particular hosts like the service provider, which may in turn access local or remote databases that are used to store and support access and search of data, but that do not define the structure through JDBC, or Java Database Connectivity, as shown for Host 1 and 3, or local queuing environments such as PBS, as shown for Host 2.

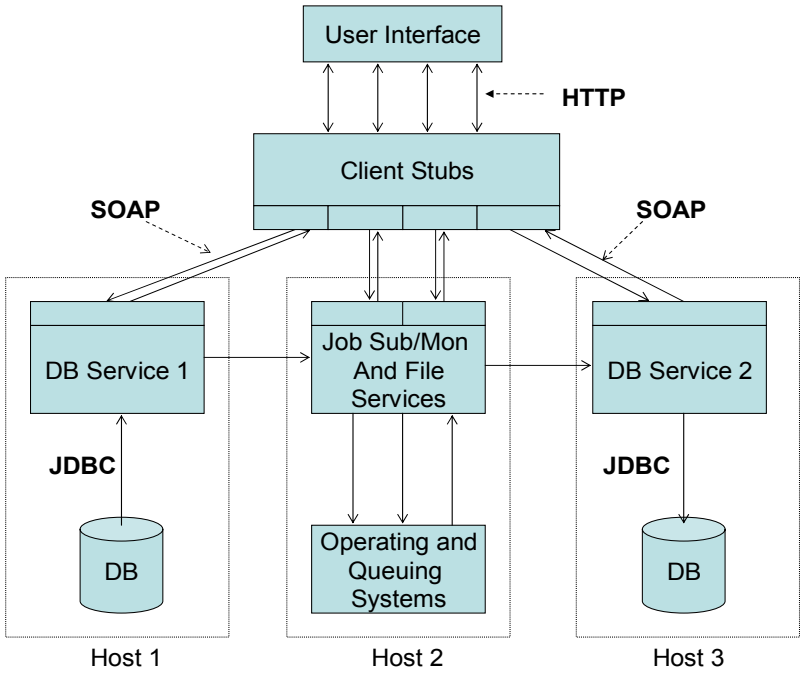


Figure 7.2: QuakeSim portal with the access of remote services

As usual, the service-oriented architecture, introduced in Chapter 4 allows the portal user to browse the database on Host 1, determine the interesting data file and transfer it to Host 2. Host 2 including a cluster or parallel computer maintains applications which may be run with the appropriate input data on a queuing system. After submitting the job, the job status is checked using the Job monitoring service. And following completion of the

job execution, the user may transfer the output back to another database or file system, or she may download the files to her desktop.

7.3 XML Descriptors for Code Input/outputs

We address services for managing input and output data formats for interactions between specified applications. A crucial problem for developing information technology-based tools for earthquake science is the definition of data structures that describe and organize the metadata associated with the field. Here it is important to distinguish between the raw data generated either by codes or by scientific instruments and the metadata that describes the raw data. The metadata is appropriately described by a specialized XML dialect. XML has the advantage of being human-readable and hierarchically organized, but is verbose and thus not ideal for very large datasets. Instead it is more often useful to have the XML metadata description point to the location of the data and describe how that data is formatted, compressed, and to be handled.

When we examined the inputs and outputs for the applications in Figure 7.1, it became apparent that the data may be split into two portions: code-independent data definitions for fault parameter and surface deformation data, and code-dependent formatting rules for incorporating the fault data and various code parameters, such as number of iterations and observation points. We therefore consider as a starting case the applications Disloc and Simplex, together with fault characterization needed by all applications.

We have designed two common XML schemas, that is, definitions of faults or surface deformations we are actually using within input and output XML formats for a particular

application such as Disloc and Simplex. Faults XML schema is available from the B.11 of the Appendix. We structure our fault definitions as being composed of the following:

- **Map View**: includes elements for longitude, latitude, and strike angle for the fault.
- **Cartesian View**: describes the location and dimensions of the fault in Cartesian space. Parameters include depth of the fault, width, length, and dip angle.
- **Material Properties**: includes various parameters needed to characterize the fault such as Lame parameters.
- **Slip**: includes the strike slip, dip slip, and tensile components of the fault slip.
- **Rate**: includes strike, dip, and tensile rates.

Surface displacements XML schema, as shown in the B.12 of the Appendix may be characterized by their locations on a two-dimensional observation plane and the values of the three dimensional displacements (or rates of displacements) and errors, including an inSAR data type, which is displacement detected in the line of sight of an inSAR satellite and has two extra column at the end, the elevation and azimuth of that satellite direction.

The input/output XML schema for Disloc and Simplex code may be viewed as B.13 and B.14 of the Appendix, respectively. These may be compared to the actual input instructions for the codes as described in [92] and [93]. Those data structures are defined in XML which has the important implication that all data is now viewed as an object. So, this XML format object may be used to generate input files for the codes in the proper legacy format, without modifying the actual source codes to take directly the XML input format. Input/output XML schemas for both codes refer the definitions of Faults and Displacements XML schema to the appropriate external schema definitions, which may be included by the use of XML namespaces. And they simply define the information

necessary to implement the input/output files. The Input/output XML schema for Disloc code, for example, defines the optional format for observation points, which may be either on a regular grid or at a group of specified points.

7.4 Implementation of services for the Data Model

In our implementations of this data model, we use Castor [65] to automatically generate Java classes equivalent to the XML Schema object specification. The generated Java language bindings for the Faults, Displacements, Input/output formats for Disloc and Simplex code XML schemas are manipulated through service implementation files. First of all, the service developer defines the well-defined API she wants to expose and wraps the corresponding Java data classes method invocations. This interface serves essentially as a façade to simplify interactions with the generated Java data classes.

We describe the service implementation using Java interfaces, which define contracts that let an implementing class know what methods it must define for compatibility. We must go a step further and define two generic interfaces that must be implemented by all applications: methods for handling Fault data and Displacement data. We also define an interface parent for code files which requires that its children implement methods for importing and exporting code data into legacy formats.

The FaultDataInterface interface fragment in Java has the following methods:

```
public interface FaultDataInterface {
  public FaultData[] getAllFaults();
  public FaultData getFault(String faultName);
  public void setFault(FaultData fault, String faultname);
```

```
    public void setAllFaults(FaultData[] fault);
  }
```

This interface defines general methods that all implementing classes must posses. And the variable FaultData (and the array of Faults, FaultData[]) are just Java bean structures of the XML Fault definition.

Like FaultDataInterface, the DisplaceDataInterface interface defines general methods for manipulating Displacements. In Java, the interface would be

```
  public interface DisplaceDataInterface {
    public DisplacementData[] getAllDisplacements();
    public void setAllDisplacements(DisplacementData[] disp);
  }
```

The variable DisplacementData is defined by the Java bean structure type for representing the XML Displacement elements.

We also require an interface parent which, when extended by a particular application code, will define how the application imports and exports the XML file to the specific input and output data legacy formats of the code. For example, we may express the input file for Disloc code using XML schema, but for generating the input file for actually running the code, we must export the XML file to the input legacy format. Similarly, when a particular application code has finished, we must import the output data and convert its legacy format into XML file, where we may for example exchange Fault or Displacement data with another application. The reverse operations must also be implemented. The GEMCodeInterface interface parent captures these requirements:

```
public interface GEMCodeIOInterface
       extends FaultDataInterface, DisplaceDataInterface {
```

```
    public void exportInputData(String xmlfile, String filename);
    public String importInputData(String filename);
    public void readData(String xmlFileName);
    public void exportOutputData(String xmlfile, String filename);
    public String importOutputData(String filename);
    public String writeData();
    public void writeData(String filename);
}
```

Those define the general method names for general import and export methods, which must be fleshed out by the implementation.

Finally, the application interface implementation must extend its interface parent, GEMCodeInterface and implement the FaultDataInterface and DisplaceDataInterface methods. It will also need to define relevant applications methods. For example, a partial listing (in Java) for Disloc code would be:

```
public interface DislocDataInterface extends GEMCodeIOInterface{
  public void setObservationStyle(int obsvStyle);
  public void setGridObsvPoints(int index, GridObsvPoints gop);
  public void setFreeObsvPoints(int index, FreeObsvPoints[] fop);
  public int getObservationStyle();
  public GridObsvPoints getGridObsvPoints(int index);
  public FreeObsvPoints[] getFreeObsvPoints(int index);
  public void setAllDislocInputData(FaultData[] fd,
                                    GridObsvPoints gop);
}
```

DislocDataInterface implementation class thus is required to implement functions for manipulating Fault and Displacement data, import and export methods that translate between XML formats and Disloc input/output legacy formats, and finally Disloc-specific methods for setting observation points, etc. The class listed above provides methods for setting the observation style and observation points for outputs, as well as material properties of the fault. The observation points for Disloc output may be either in a regular grid or on specified surface points. And those "set/getGridObsvPoints" and "set/getFreeObsvPoints" methods may be used to access the appropriate output of the surface deformations. Similarly, Simplex code may be also implemented except defining the Simplex-specific methods such as Free/Fixed flag for Fault parameter, number of iteration and starting temperature point.

The above code fragment is next converted into WSDL for the client shown in the A.8 of the Appendix for the "Disloc" code's data service and the A.9 of the Appendix for the "Simplex" code's data service, and then may be used by client developers to create methods for remote service invocation. The value of WSDL and SOAP here is again evident: the FaultData, DispalcementData class, and etc, which are following Java bean structure, for example, may be directly cast back into its XML form, that is WSDL complex type and sent in a SOAP message to the remote service.

7.5 Data Service Architecture

We need to define the architecture that describes how the various services must interact with each other. We first consider the case for "Disloc", one of applications which are running, as illustrated in Figure 7.3. For this process, all of services which are

needed for are deployed into distributed hosts and described in WSDL for the client, and SOAP is used for inter-component communication.

The browser interface is used to gather user code selections, for example, “Disloc” and a desired host where “Disloc” code may be executed. The user then fills out HTML forms providing the information needed to construct the Faults and Input/output for Disloc code XML schemas. These pages are created and managed by the User Interface Server, which also acts as a control point (through client stubs) for managing the remote services. These HTTP requests are translated into XML encoding by the “Disloc” data Web service, which implements the interfaces described previously. This file is then exported to the legacy format and transferred to the execution host.

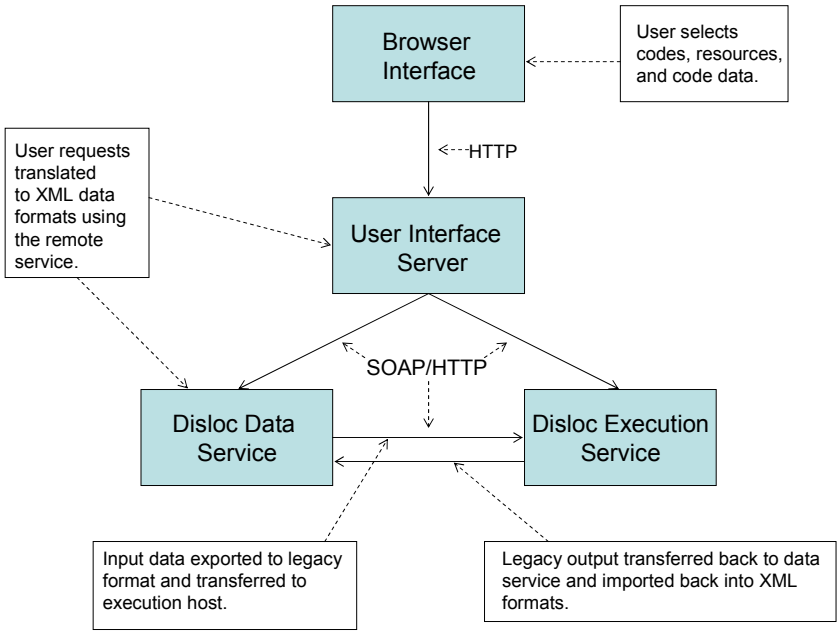


Figure 7.3: Interactions of the Disloc data service. Shaded boxes indicate distributed components. Solid arrow lines indicate over-the-wire transmissions with the indicated protocols.

There are two cases for the completion of the code's execution according to host environments. One is to execute the application code directly on local or remote host. When the code exits, the legacy data format may be transferred back to the Data Service and imported into XML. The other, as shown in Figure 7.4 is to run the application code through the queuing system such as PBS server on local or remote host. PBS has email notification system about the job abortion, the job beginning, and the job end. When the job is done, PBS sends the user an email which includes the information about the job completion.

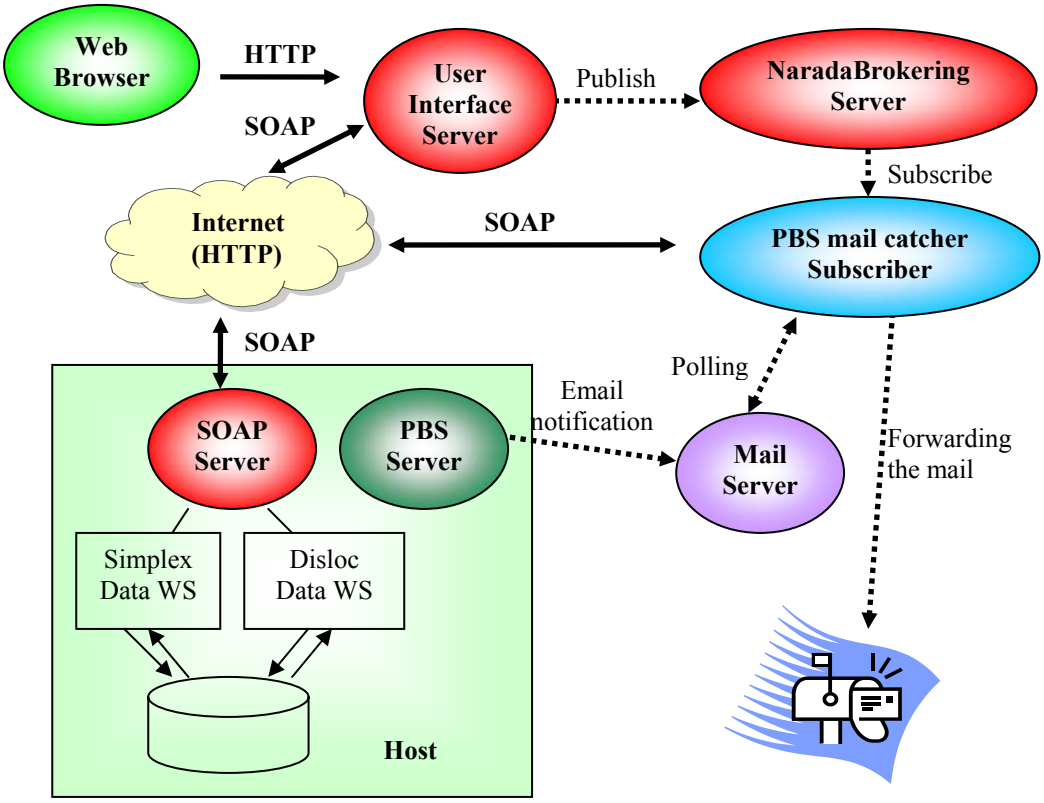


Figure 7.4: Conversion the legacy output data into XML format using the email notification in the PBS queuing system when the job is done.

Using that email notification, we have implemented the PBS email catcher, acting as the subscriber through the NaradaBrokering server [82], which is Java Message Service

(JMS) compliance that is publish/subscribe model based on messaging system. When the job is submitted, the job ID from the PBS server is returned to the User Interface Server. And then in order to activate the PBS email catcher, the User Interface Server publishes the job ID and user's private email into the NaradaBrokering server. The PBS email catcher which is a subscriber polls then the mail server periodically, comparing with the job ID. Instead of using the user's private email, we used the public email account, for example, the project account for easily checking the job status and protecting the user's mailbox. When the email notification about the completion of the job is arrived, the PBS email catcher forwards it to user's private email and then sends the application legacy output file into Data service for importing into XML format.

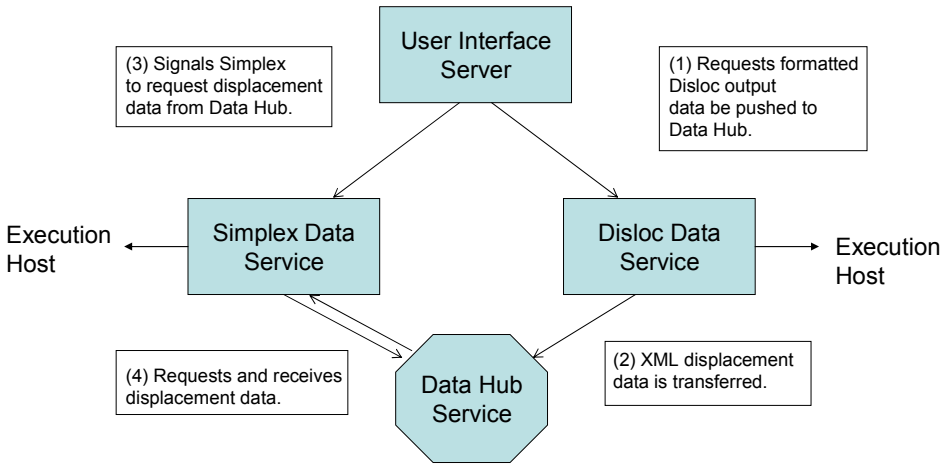


Figure 7.5: Simplex and Disloc code share data through the Data Hub service. Shaded objects represent distributed components. Closed arrows represent connections with

execution services such as shown in Figure 7.3. The open arrows represent SOAP over HTTP invocations.

The output data is allowed to be shared in a common data format, which refers to the raw data to be manipulated, with other applications [94]. Visualization and analysis services and an application such as Simplex code may acquire the displacement data from the output data generated by "Disloc" code. Such data sharing is enabled by the common XML data format, and is performed through the Data Hub service for extracting the Fault and Displacement data from the formatted XML output of applications, as illustrated in Figure 7.5.

We assume the interactions of Figure 7.3 have already taken place for describing Figure 7.5. So, the User Interface Server again acts as the central control point and issues commands (at either user request or through automating events) initially to the "Disloc" data service to transfer its output data to the hub (Step 1). The service accomplishes this in Step 2. In Step 3, the User Interface Server requests that Simplex code should import selected displacements, which are already generated from "Disloc" code, from the Data Hub. The data in XML format is then imported in Step 4. The "Simplex" Data Service may generate a "Simplex" legacy input file with additional Fault data and Simplex-specified parameters and execute the Simplex application code, as shown in Figure 7.3.

7.6 QuakeSim Portal Toolkits

We developed the web-based computing portal system, SERVOGrid Interoperability framework for the user interaction with the remote services, as described the previous sections. It aggregates access to distributed services which are needed for the GEM

simulation, analysis and visualization. User interfaces to GEM services (Code Submission, Job Monitoring, File Management for Host X) are all managed as portlets which provide component model for user interfaces. And Users, administrators can customize their portal interfaces to just precisely the services they want.

7.6.1 The user interface for the code submission

We have developed a toolkit that allows one to build a suite of general purpose Grid Web services for managing distributed applications. Using the browser-based user interface, users can choose applications, submit jobs for executing applications and analyze outputs, as shown in Figure 7.6 – 7.10. This process consists of several major steps.

Figure 7.6: Defining tracks for the application selection to create new sessions and for the problem archive of old sessions.

First, QuakeSim portal aggregates portlets into a single, user-customizable display, using Jetspeed framework which is a free, open source portlet implementation. Tabs such as “GEM Application”, “Noahsark”, “Solar”, and so on shown in Figure 7.6 indicate

available portlet interfaces. “GEM applications” portlet interface provides seamlessly access to distributed applications.

Second, the code selection track shown in Figure 7.7 allows users to choose GEM applications and hosts for running jobs. This page is dynamically created from the application descriptor XML data records of available codes and machines. This application metadata record can be edited through the application administration portlet interface (Step 1). In Step 2, the process of making input data for that selected application is chosen to get the data from the Data Hub or to create new data. If the user selects the Data Hub option, user’s generated GEM data sessions are displayed in Step 3.

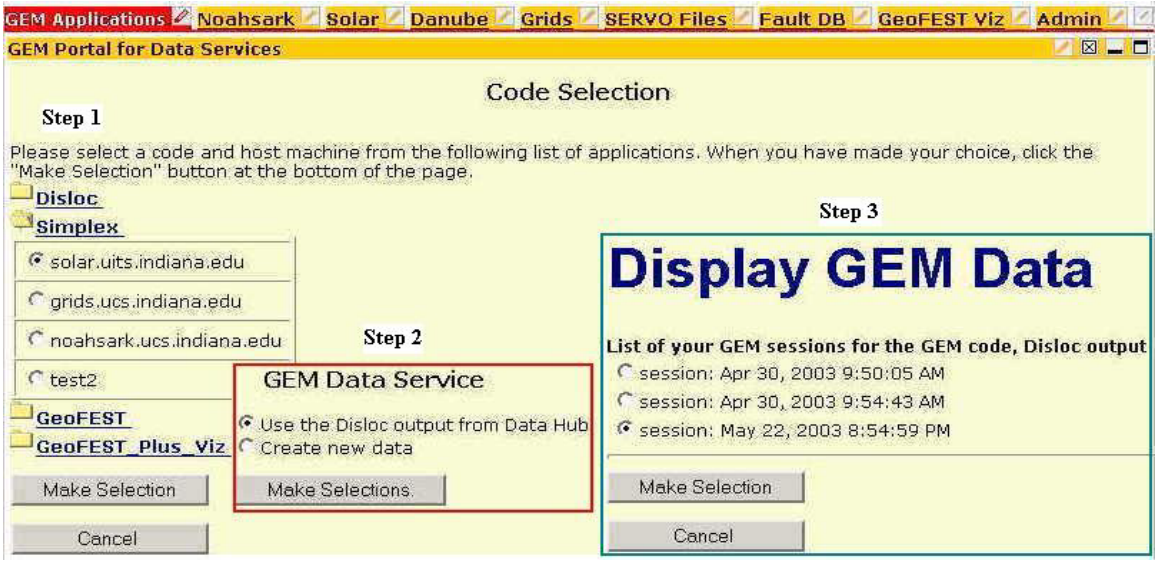


Figure 7.7: Selecting a GEM code for simulating and a host for running (Step 1), input data format which is used for the code (Step 2) and displaying the GEM data in user’s session (Step 3).

Third, this input data process for the selected application shown in Figure 7.8 is displayed with the Faults data from the Data Hub and application-specific data provided by HTML forms. This page also includes the Displacements data from the Data Hub

which is not displayed for generating the selected application input data. We think that displaying very huge data is not important in this page.

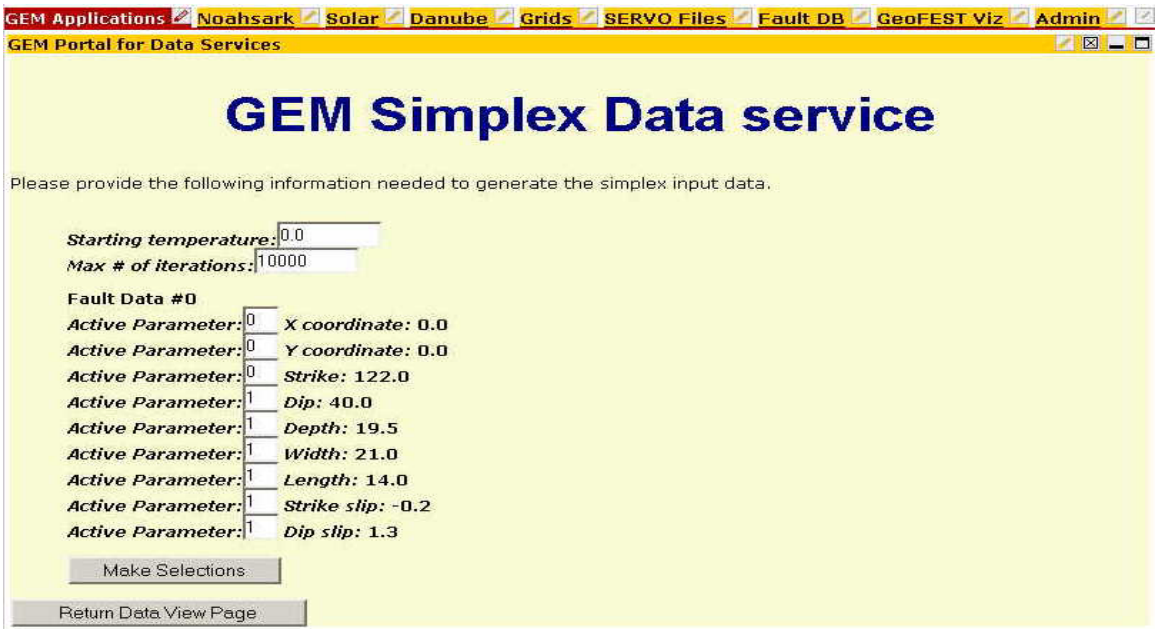


Figure 7.8: Getting the displacement and fault data from the Data Hub for generating the legacy input data of the Simplex code.

Fourth, the job script generator page, shown in Figure 7.9, illustrates a form generated for the user for submitting a particular application (a finite element code, in this case) to the indicated queuing system for example, PBS, based on an Application Descriptor instance and the output location of the selected application. This page also generates the legacy input format for that application, saves it into the appropriate input location on selected host and put its location into the application input field provided by HTML form automatically.

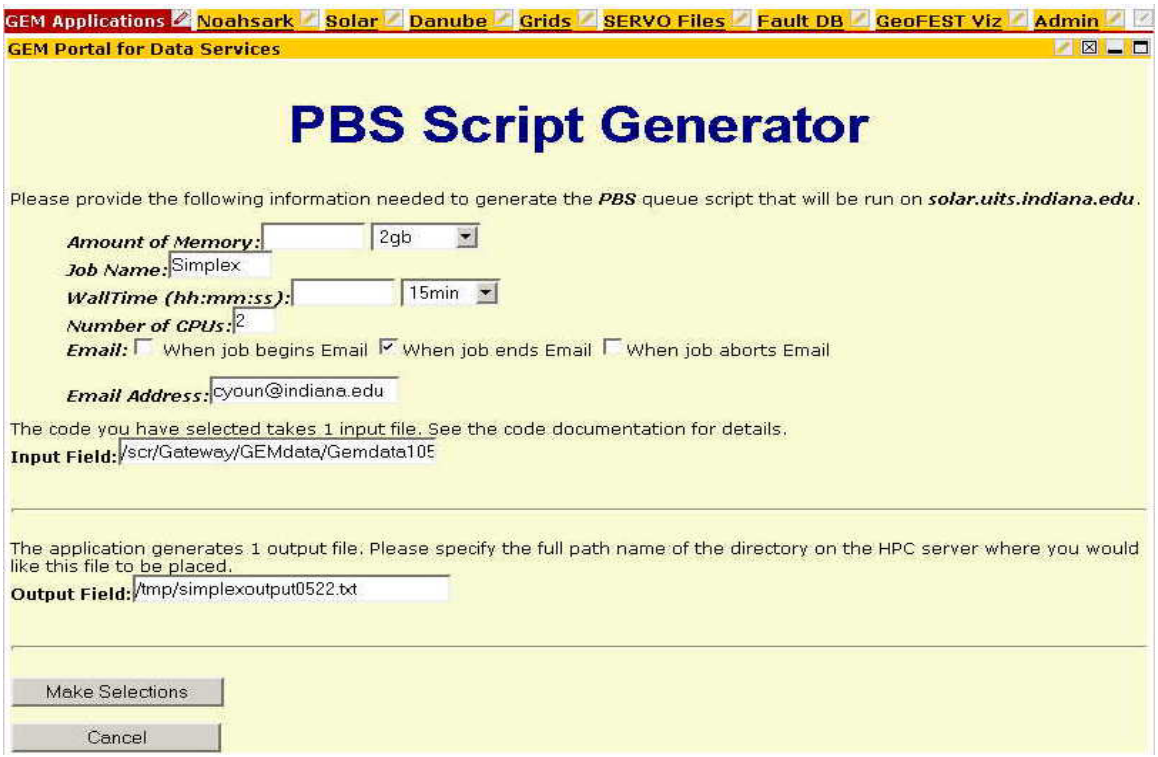


Figure 7.9: Generating the script for the job submission.

Fifth, the job script reviewing page shown in Figure 7.10 allows users to view the job script before submitting the job. This page also generates the application instance XML file which includes all of user's requests, save it into the user's session directory for retrieving later in "View Archive" section shown in Figure 7.6 and then creates the user script for running the selected application, extracting the data from the application instance XML format

Finally, the user's job script is saved into the user's session directory and is transferred if the User Interface Server is different with the selected host.

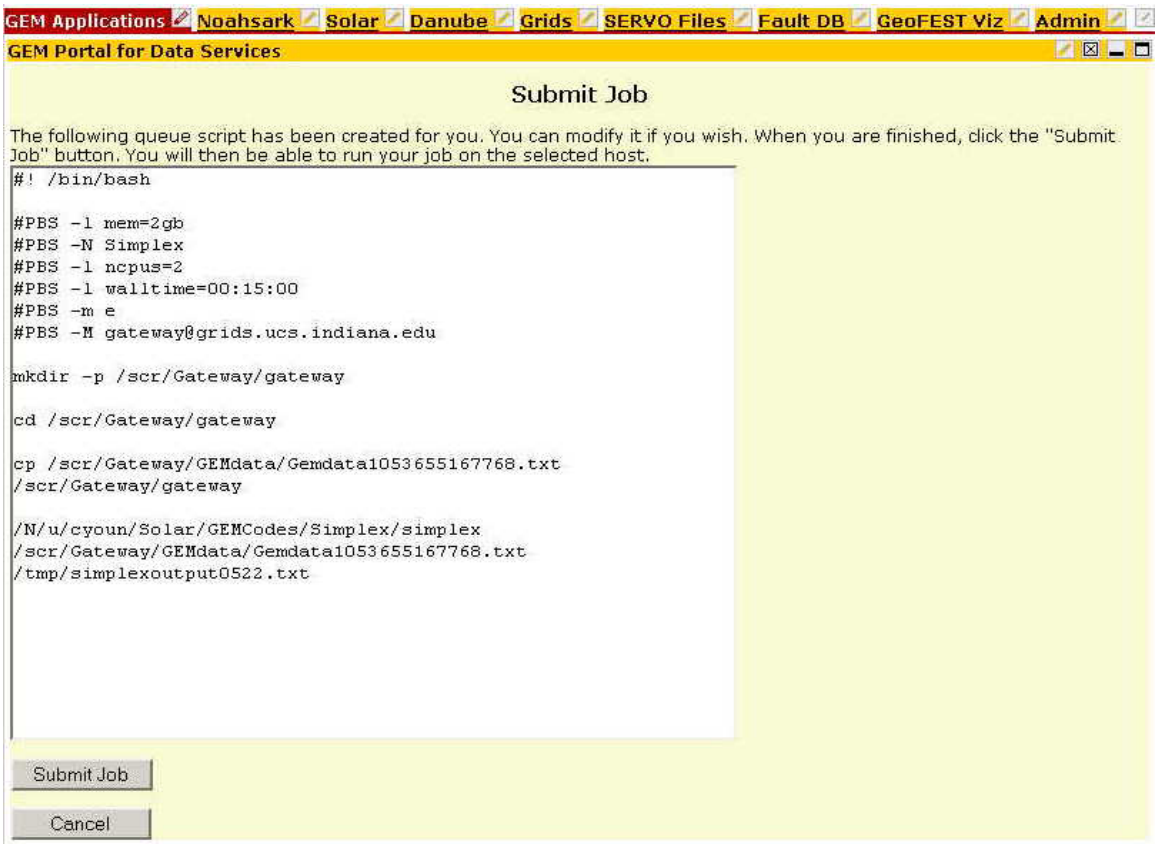


Figure 7.10: Reviewing the generated job script before submitting the job.

7.6.2 The user interface for File Management and Job Monitoring

The file management service shown in Figure 7.11 allows users to view files to be handled on selected remote host. File operation includes Upload, Download, Copy, Rename, Change Directory, and Crossroad. These can be done by using simple point-and-click interface to get the information about applications for building the job script. Files can be uploaded or downloaded between user's desktop and selected remote host, and may be crossloaded between two remote hosts.

From the previous section 7.6.1, we built the job script for executing the application code and submitted the job into queuing system such as PBS on a specific host. If users

want to know their current job status on that host, the Job Monitoring service shown in Figure 7.11 is allowed to retrieve user's job status.

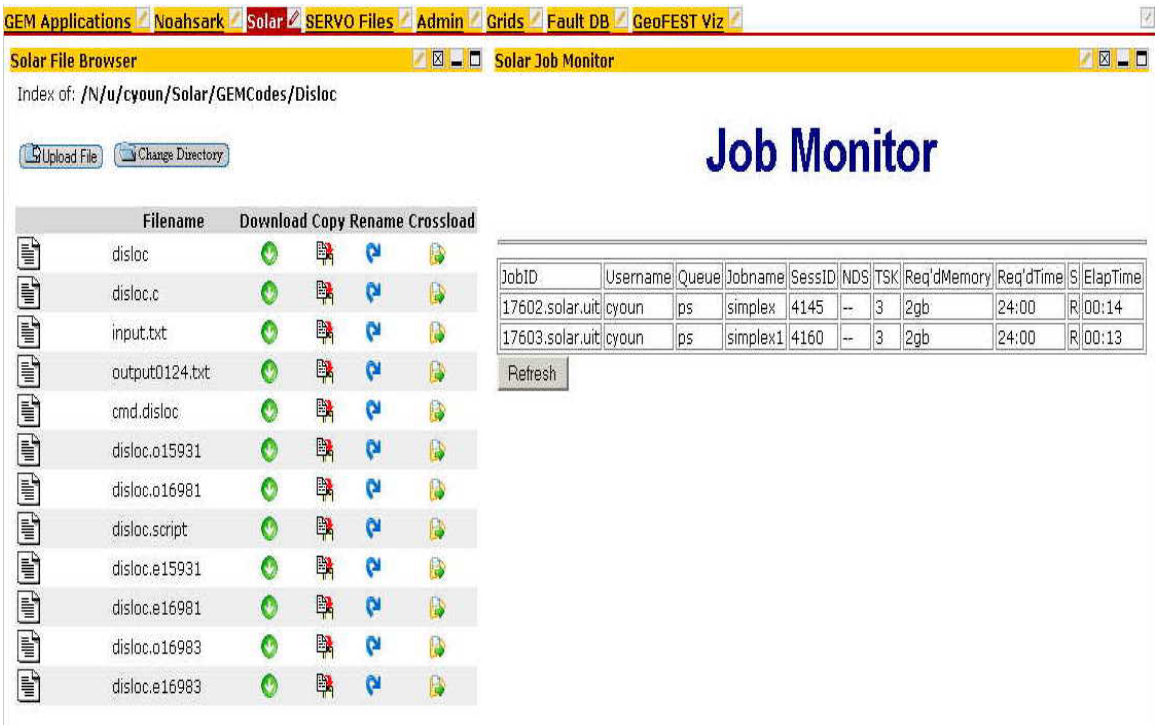


Figure 7.11: File Browser and Job Monitoring user interface.

7.7 Summary

We have presented our initial architecture for implementing code specific data services, and then developed the QuakeSim portal toolkits which are the code submission, the file manipulation, and the job monitoring. This consists of three major steps. First, we must examine the relevant applications and their input and output files to extract out the code-independent data from the code-dependent instructions. We then express all of these with XML schema. In particular, we have developed application independent schemas for faults and displacements, as well as code specific schemas for "Disloc" and "Simplex" input and output data, two of the applications shown in Figure 7.1.

Next, we must wrap these data models in useful services that can be plugged into our Web service-based portal. These modules must implement a set of specified interfaces for manipulating Faults and Displacements data, as well as a parent interface that requires the service to implement import and export functions for converting between XML and legacy input/output formats. These services may then be deployed and clients built following normal Web service development patterns.

Finally, we must provide a means for connecting two code-specific data services. This is done by a central or distributed data hub that can import and export XML-encoded fault and displacement data. This can be used to share, for example, synthetic "Disloc" displacement data with "Simplex".

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this dissertation we have presented a comprehensive view of computing portal architectures based around Web Services approach, forming the service-oriented computing architecture. It is standard-based and follows a distributed XML messaging paradigm, facilitating a seamless integration of commodity software component. This is our effort with a description of the initial investigations into the core services, interoperability issues, and security that will form the basis for an interoperable and interchangeable Web Service architecture for computational portals. Going beyond these services, we discussed the need for application-specific services and data models that can be used to encapsulate entire applications in portal implementation independent way.

We have emphasized on the development of reusable services that can form the basis for multiple Problem Solving Environments. We have identified and classified the kinds of services depending on the service deployment. Typical basic portal services provided by these portals include job submission, job monitoring, file transfer, context manager, script generation, and Application Web Services. They can be grouped into two categories which we define: Host-specific service and Host-independent service. Host-specific services are allowed users to run the user's applications from the hosting environment directly. So, they should be deployed on the each specific host for users:

- Job monitoring service supports the run access to get user's job status from the queuing system running on the specific host.
- Job submission service supports the run access to submit jobs on the specific host that manages HPC resources such as super computers, and sun clusters.
- File Transfer service supports uploading, downloading, and file manipulation such as read file, copy, and rename on specific host which users manages the input and output data of their scientific application codes and files.

Host-independent services are related to user's metadata that are best realized in XML, which provides a relatively simple, programming-language neutral approach and are geared to particular applications that handle the details of where and when jobs get run. They are very informational services for users. Regardless of the host-specific location, they can be accessed anywhere:

- Context manager service is for capturing and organizing the user's session (or context) for archival purposes.

- Script Generation service is for creating user's job script which consists of batch queue script such as PBS, GRD and the UNIX shell commands and the executable information of the scientific application code.
- Application Web Service is for wrapping the application codes into services that is applicable to any portal designed to support multiple codes.

Based on this classification, the portal developer can construct specific implementations and composites of primitive service components and can also provide services that may be shared among different portals. Legacy applications and services can be 'wrapped' to make them Web services that are available to appropriate users, including other portals and applications. We have experience with the SDSC HotPage and IU Gateway portals, and by following the Web services approach, we are able to share existing services that we previously developed independently.

As part of a first test bed for evaluating interoperability and reusability, we chose to build a simple, anonymous, Batch Script Generator Web Service, and have explored using a directory service (UDDI) to publish information about the Web service. We have shown that this Web service can be used by multiple portals. This work has demonstrated to us that the conversion of existing portal functions to Web services mechanisms and adherence to emerging Web services standards in development of future grid application can be done and will ease the task of portal developers and provide new levels of flexibility and power to portal users. Many decisions about specific implementation details for using Web services for computational portals remain to be made, and through participation in the GCE Research Group of the Global Grid Forum and other community efforts, we will play an active role in determining the evolution of portals and Web

services.

We have developed QuakeSim portal for the earthquake science with an application toolkit implemented in the front end (portlet interface) integrated with the networked and distributed core Web services. We have also presented the code-specific data service architecture with sharing the application real data through the data hub. There are two possible future revisions in our architecture. The first is in data encoding. XML is a verbose way for marking up data, and an alternate approach may be to encode only metadata in XML and use an alternative data format, such as Hierarchical Data Format (HDF) which is software and file formats for scientific data management, developed by NCSA [95], for the large data sets. We will need to base this on network and host performance tests for a large number of use cases. The second possible modification is in the nature of the Data Hub. As shown in Figure 7.5, we have designed this to be a Web service component and assume point-to-point messaging. However, we must also explore alternative publish/subscribe mechanisms that will allow subscribed hosts (such as "Simplex" in Figure 7.5) to be notified when interesting data has been published. This mechanism would remove some of the low-level control capabilities from the User Interface server.

We have considered some specific extensions (security, negotiation) to the architecture of a Web Service based computing portal, including the job composition as the further study. First, currently our architecture of Gateway allows us to put the security into SOAP messages between user interface server and the repository and the service provider for the user authentication, based on the message-level security architecture. SOAP security can be provided through standard interfaces to independently specific

mechanisms such as Kerberos, PKI, and GSI. Our general approach is used for putting the assertion based security such as SAML, WS-security into SOAP messages. Next, we have provided the prototype system for the service compatibility depending on their computing environments. As of examples, we implemented a specific application, the version control example which is a particular computing service, job submission service version.

8.2 Future Work

This research has presented a service-oriented framework for computing portals. Still, we have a great deal of research issues left open to discuss. In the following we identify several cases for our future research work.

8.2.1 Use of service architecture with proxy-style portal frontended by aggregation portal

We believe that these atomic computing services discussed in this dissertation, while independently interesting, need to be integrated into a common architecture. Figure 8.1 schematically illustrates a possible architecture that illustrates these ideas. The integrated architecture can be configured like a distributed operating system on the web computing environments. In Unix system, user interactions are through a finite list of basic commands that operate in a “shell” or execution environment. These commands encapsulate “system” level calls to actually interact with computing resources. The point is that there are two levels of interfaces: one to the system level Grid infrastructure and

one to the core portal service. The user does not interact with the system level services but instead through a tool chest of core services, which in turn interact with the system level interfaces.

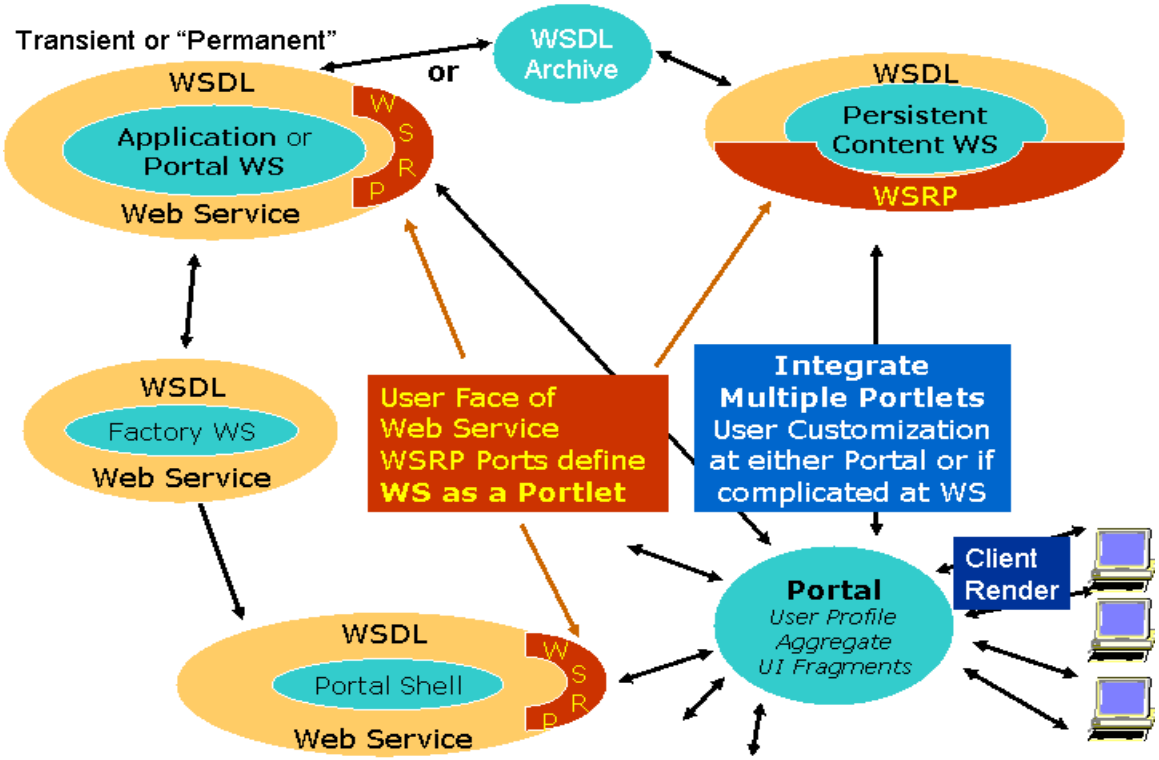


Figure 8.1: A schematically comprehensive service-based portal architecture.

Figure 8.1 consists of three types of portal Web Services such as applications, portal shell commands, and content (information) services that interact with XML data objects. Each has a WSDL (or perhaps OGSA) definition and is implemented out of system level Grid services. The portal shell services are basic services which we have described in Chapter 4. These services may be bound to specific resources through a factory creation process, such as discussed in Ref. [96]. One may envision a scripting environment for example that provides the syntax for linking the various core services (redirecting output through pipes, for example) and the logic for executing services. The Application Web Services discussed in Chapter 5 represent one way of aggregating core services for the

specific purpose of describing scientific applications. Finally, all of the portal services (core, application, information, and others) define interfaces that separate the service implementation from the client implementation. The client implementation is created from the published interface and may either be static or dynamically generated using the automatic user interface generation system [97] which is a general purpose system for creating user interfaces and action bindings for a particular XML schema. These client interfaces themselves can be aggregated into a portal interface. The discovery, binding, and communication between such portlet components may be handled through the Web Services for Remote Portlets (WSRP) [42] standards for "Pluggable" integration with all portals. Through WSRP, this process could be simplified to integrate with other portals.

8.2.2 Particular services needed

Now we produce a summary table of "services" including all of computing services presented in this dissertation for using some taxonomy.

Area	Service
Globus/OGSA level	Job Submission
	Job Monitoring
	File Manipulation
	Context Management
	Script Generation
Workflow Application level	Application Web Service
	PSE services
Jetspeed (portlet) level	Aggregation Portal services

Table 8.1: Summary of Services

In the Table 8.1 the basic computing services which we are defined as the Globus/OGSA level are introduced in Chapter 4. Aggregation portal services in the portlet level are explained in Section 8.2.1. Finally, in the workflow application level, the

Web services can be thought of as atomic components that can be used to compose complicated service. This entire job can be considered to be a single, composite job that is an application-centric service. Therefore, we need the further study for the workflow for handling the Web services using Web Services Flow Language (WSFL) [64], and our own XML dialects which we will define metadata and write components for handling them. WSFL provided by IBM Software Group is XML-based language for the description of Web Services compositions, consisting of flow model that describes how to use the functionality provided by the collection of composed Web Services and global model which provides a description of how the composed Web Services interact with each other.

8.2.3 Issues connected to security with different needs in different cases

We have pursued a general purpose way of securing SOAP messages that support multiple underlying mechanisms independently based on user assertions. The prototype system using Kerberos and SAML has been done for this. However, major computing centers and other institutions apply different security technologies such as SSH, SSL, Kerberos, or other. So, a scheme should be provided to ensure secure communications among authorized resources in distributed organizations. This scheme should also incorporate strategies to detect a security compromise among organizations. For example, in the message-level security framework, the message may be encrypted differently by one organization, allowing users with access resources of the other organization. Those basic security operations such as authentication and authorization process should be

performed in a mechanism-independent way, with specific mechanisms such Kerberos, PKI and GSI, plugged into some particular applications.

Appendix A

WSDL (Web Services Description Language)

A.1 Job submission service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob"
xmlns:intf="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob"
xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_xsd_string">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
   </restriction>
   </complexContent>
   </complexType>
   <element name="ArrayOf_xsd_string" nillable="true" type="impl:ArrayOf_xsd_string" />
   </schema>
   </wsdl:types>
 <wsdl:message name="execLocalCommandResponse">
   <wsdl:part name="execLocalCommandReturn" type="intf:ArrayOf_xsd_string" />
   </wsdl:message>
 <wsdl:message name="testResponse">
   <wsdl:part name="testReturn" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="execRemoteCommandResponse">
```

```
  <wsdl:part name="execRemoteCommandReturn" type="intf:ArrayOf_xsd_string" />
  </wsdl:message>
 <wsdl:message name="execRemoteCommandRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  </wsdl:message>
 <wsdl:message name="execLocalCommandRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="testRequest" />
 <wsdl:portType name="SJwsImp">
 <wsdl:operation name="execLocalCommand" parameterOrder="in0">
  <wsdl:input message="intf:execLocalCommandRequest" name="execLocalCommandRequest" />
  <wsdl:output message="intf:execLocalCommandResponse" name="execLocalCommandResponse" />
  </wsdl:operation>
 <wsdl:operation name="test">
  <wsdl:input message="intf:testRequest" name="testRequest" />
  <wsdl:output message="intf:testResponse" name="testResponse" />
  </wsdl:operation>
 <wsdl:operation name="execRemoteCommand" parameterOrder="in0 in1 in2 in3">
  <wsdl:input message="intf:execRemoteCommandRequest" name="execRemoteCommandRequest" />
  <wsdl:output message="intf:execRemoteCommandResponse" name="execRemoteCommandResponse" />
  </wsdl:operation>
  </wsdl:portType>
 <wsdl:binding name="SubmitjobSoapBinding" type="intf:SJwsImp">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="execLocalCommand">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="execLocalCommandRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:input>
 <wsdl:output name="execLocalCommandResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="test">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="testRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:input>
 <wsdl:output name="testResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="execRemoteCommand">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="execRemoteCommandRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:input>
 <wsdl:output name="execRemoteCommandResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="SJwsImpService">
 <wsdl:port binding="intf:SubmitjobSoapBinding" name="Submitjob">
  <wsdlsoap:address location="http://solar.uits.indiana.edu:8005/GCWS/services/Submitjob" />
  </wsdl:port>
  </wsdl:service>
```

```
</wsdl:definitions>
```

A.2 File Manipulation service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://solar.uits.indiana.edu:8005/GCWS/services/FileService"
xmlns:intf="http://solar.uits.indiana.edu:8005/GCWS/services/FileService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="FileService"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService"
xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_xsd_string">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
   </restriction>
   </complexContent>
   </complexType>
   <element name="ArrayOf_xsd_string" nillable="true" type="impl:ArrayOf_xsd_string" />
   </schema>
 <schema targetNamespace="FileService" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
   <element name="DataHandler" nillable="true" type="tns1:DataHandler" />
   </schema>
   </wsdl:types>
 <wsdl:message name="uploadFileResponse">
   <wsdl:part name="uploadFileReturn" type="xsd:boolean" />
   </wsdl:message>
 <wsdl:message name="copyFileRequest">
   <wsdl:part name="sourceFile" type="xsd:string" />
   <wsdl:part name="destFile" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="listDirectoriesResponse">
   <wsdl:part name="listDirectoriesReturn" type="intf:ArrayOf_xsd_string" />
   </wsdl:message>
 <wsdl:message name="listFilesResponse">
   <wsdl:part name="listFilesReturn" type="intf:ArrayOf_xsd_string" />
   </wsdl:message>
 <wsdl:message name="downloadFileRequest">
   <wsdl:part name="serverFileName" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="renameFileResponse">
   <wsdl:part name="renameFileReturn" type="xsd:boolean" />
   </wsdl:message>
 <wsdl:message name="crossloadRequest">
   <wsdl:part name="sourceFile" type="xsd:string" />
   <wsdl:part name="destServiceName" type="xsd:string" />
   <wsdl:part name="destFile" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="listFilesRequest">
   <wsdl:part name="dirPath" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="crossloadResponse">
   <wsdl:part name="crossloadReturn" type="xsd:boolean" />
   </wsdl:message>
 <wsdl:message name="downloadFileResponse">
   <wsdl:part name="downloadFileReturn" type="tns1:DataHandler" />
   </wsdl:message>
```

```
<wsdl:message name="renameFileRequest">
  <wsdl:part name="oldname" type="xsd:string" />
  <wsdl:part name="newname" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="uploadFileRequest">
  <wsdl:part name="dh" type="tns1:DataHandler" />
  <wsdl:part name="serverFileName" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="copyFileResponse">
  <wsdl:part name="copyFileReturn" type="xsd:boolean" />
  </wsdl:message>
<wsdl:message name="listDirectoriesRequest">
  <wsdl:part name="dirPath" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="FileServiceImp">
<wsdl:operation name="listFiles" parameterOrder="dirPath">
  <wsdl:input message="intf:listFilesRequest" name="listFilesRequest" />
  <wsdl:output message="intf:listFilesResponse" name="listFilesResponse" />
  </wsdl:operation>
<wsdl:operation name="listDirectories" parameterOrder="dirPath">
  <wsdl:input message="intf:listDirectoriesRequest" name="listDirectoriesRequest" />
  <wsdl:output message="intf:listDirectoriesResponse" name="listDirectoriesResponse" />
  </wsdl:operation>
<wsdl:operation name="renameFile" parameterOrder="oldname newname">
  <wsdl:input message="intf:renameFileRequest" name="renameFileRequest" />
  <wsdl:output message="intf:renameFileResponse" name="renameFileResponse" />
  </wsdl:operation>
<wsdl:operation name="copyFile" parameterOrder="sourceFile destFile">
  <wsdl:input message="intf:copyFileRequest" name="copyFileRequest" />
  <wsdl:output message="intf:copyFileResponse" name="copyFileResponse" />
  </wsdl:operation>
<wsdl:operation name="crossload" parameterOrder="sourceFile destServiceName destFile">
  <wsdl:input message="intf:crossloadRequest" name="crossloadRequest" />
  <wsdl:output message="intf:crossloadResponse" name="crossloadResponse" />
  </wsdl:operation>
<wsdl:operation name="uploadFile" parameterOrder="dh serverFileName">
  <wsdl:input message="intf:uploadFileRequest" name="uploadFileRequest" />
  <wsdl:output message="intf:uploadFileResponse" name="uploadFileResponse" />
  </wsdl:operation>
<wsdl:operation name="downloadFile" parameterOrder="serverFileName">
  <wsdl:input message="intf:downloadFileRequest" name="downloadFileRequest" />
  <wsdl:output message="intf:downloadFileResponse" name="downloadFileResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="FileServiceSoapBinding" type="intf:FileServiceImp">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="listFiles">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="listFilesRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
<wsdl:output name="listFilesResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="listDirectories">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="listDirectoriesRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
<wsdl:output name="listDirectoriesResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
```

```
  </wsdl:operation>
 <wsdl:operation name="renameFile">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="renameFileRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
 <wsdl:output name="renameFileResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="copyFile">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="copyFileRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
 <wsdl:output name="copyFileResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="crossload">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="crossloadRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
 <wsdl:output name="crossloadResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="uploadFile">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="uploadFileRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
 <wsdl:output name="uploadFileResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="downloadFile">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="downloadFileRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:input>
 <wsdl:output name="downloadFileResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="FileServiceImpService">
 <wsdl:port binding="intf:FileServiceSoapBinding" name="FileService">
  <wsdlsoap:address location="http://solar.uits.indiana.edu:8005/GCWS/services/FileService" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.3 Job monitor service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor"
xmlns:intf="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="JobObjectResult">
 <sequence>
   <element name="elapTime" nillable="true" type="xsd:string" />
   <element name="jobID" nillable="true" type="xsd:string" />
   <element name="jobname" nillable="true" type="xsd:string" />
   <element name="nds" nillable="true" type="xsd:string" />
   <element name="queue" nillable="true" type="xsd:string" />
   <element name="reqdMemory" nillable="true" type="xsd:string" />
   <element name="reqdTime" nillable="true" type="xsd:string" />
   <element name="sessID" nillable="true" type="xsd:string" />
   <element name="status" nillable="true" type="xsd:string" />
   <element name="tsk" nillable="true" type="xsd:string" />
   <element name="username" nillable="true" type="xsd:string" />
   </sequence>
   </complexType>
 <complexType name="ArrayOfJobObjectResult">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:JobObjectResult[]" />
   </restriction>
   </complexContent>
   </complexType>
   <element name="ArrayOfJobObjectResult" nillable="true" type="tns1:ArrayOfJobObjectResult" />
   </schema>
   </wsdl:types>
 <wsdl:message name="JobStatusResponse">
   <wsdl:part name="JobStatusReturn" type="tns1:ArrayOfJobObjectResult" />
   </wsdl:message>
 <wsdl:message name="JobStatusRequest">
   <wsdl:part name="in0" type="xsd:string" />
   <wsdl:part name="in1" type="xsd:string" />
   </wsdl:message>
 <wsdl:portType name="JMwsImp">
 <wsdl:operation name="JobStatus" parameterOrder="in0 in1">
   <wsdl:input message="intf:JobStatusRequest" name="JobStatusRequest" />
   <wsdl:output message="intf:JobStatusResponse" name="JobStatusResponse" />
   </wsdl:operation>
   </wsdl:portType>
 <wsdl:binding name="JobmonitorSoapBinding" type="intf:JMwsImp">
   <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="JobStatus">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="JobStatusRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor" use="encoded" />
   </wsdl:input>
 <wsdl:output name="JobStatusResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
   </wsdl:binding>
```

```
<wsdl:service name="JMwsImpService">
<wsdl:port binding="intf:JobmonitorSoapBinding" name="Jobmonitor">
  <wsdlsoap:address location="http://solar.uits.indiana.edu:8005/GCWS/services/Jobmonitor" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.4 Context manager service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager"
xmlns:intf="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_soapenc_string">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
   </restriction>
   </complexContent>
   </complexType>
   <element name="ArrayOf_soapenc_string" nillable="true" type="tns1:ArrayOf_soapenc_string" />
   </schema>
   </wsdl:types>
 <wsdl:message name="getCurrentPropertyResponse">
   <wsdl:part name="getCurrentPropertyReturn" type="xsd:string" />
   </wsdl:message>
   <wsdl:message name="addContextResponse" />
   <wsdl:message name="setContextStorageResponse" />
 <wsdl:message name="initRequest">
   <wsdl:part name="userName" type="xsd:string" />
   <wsdl:part name="descDir" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="listContextRequest">
   <wsdl:part name="contexts" type="xsd:string" />
   </wsdl:message>
   <wsdl:message name="setCurrentPropertyResponse" />
   <wsdl:message name="removeContextResponse" />
 <wsdl:message name="setCurrentPropertyRequest">
   <wsdl:part name="contexts" type="xsd:string" />
   <wsdl:part name="propName" type="xsd:string" />
   <wsdl:part name="propValue" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="listContextResponse">
   <wsdl:part name="listContextReturn" type="tns1:ArrayOf_soapenc_string" />
   </wsdl:message>
 <wsdl:message name="addContextRequest">
   <wsdl:part name="contextpath" type="xsd:string" />
   </wsdl:message>
 <wsdl:message name="setContextStorageRequest">
   <wsdl:part name="storage" type="xsd:string" />
   </wsdl:message>
   <wsdl:message name="initResponse" />
 <wsdl:message name="getCurrentPropertyRequest">
   <wsdl:part name="contexts" type="xsd:string" />
   <wsdl:part name="propname" type="xsd:string" />
   </wsdl:message>
```

```
<wsdl:message name="removeContextRequest">
  <wsdl:part name="contexts" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="ContextManagerImp">
<wsdl:operation name="init" parameterOrder="userName descDir">
  <wsdl:input message="intf:initRequest" name="initRequest" />
  <wsdl:output message="intf:initResponse" name="initResponse" />
  </wsdl:operation>
<wsdl:operation name="removeContext" parameterOrder="contexts">
  <wsdl:input message="intf:removeContextRequest" name="removeContextRequest" />
  <wsdl:output message="intf:removeContextResponse" name="removeContextResponse" />
  </wsdl:operation>
<wsdl:operation name="setContextStorage" parameterOrder="storage">
  <wsdl:input message="intf:setContextStorageRequest" name="setContextStorageRequest" />
  <wsdl:output message="intf:setContextStorageResponse" name="setContextStorageResponse" />
  </wsdl:operation>
<wsdl:operation name="addContext" parameterOrder="contextpath">
  <wsdl:input message="intf:addContextRequest" name="addContextRequest" />
  <wsdl:output message="intf:addContextResponse" name="addContextResponse" />
  </wsdl:operation>
<wsdl:operation name="getCurrentProperty" parameterOrder="contexts propname">
  <wsdl:input message="intf:getCurrentPropertyRequest" name="getCurrentPropertyRequest" />
  <wsdl:output message="intf:getCurrentPropertyResponse" name="getCurrentPropertyResponse" />
  </wsdl:operation>
<wsdl:operation name="setCurrentProperty" parameterOrder="contexts propName propValue">
  <wsdl:input message="intf:setCurrentPropertyRequest" name="setCurrentPropertyRequest" />
  <wsdl:output message="intf:setCurrentPropertyResponse" name="setCurrentPropertyResponse" />
  </wsdl:operation>
<wsdl:operation name="listContext" parameterOrder="contexts">
  <wsdl:input message="intf:listContextRequest" name="listContextRequest" />
  <wsdl:output message="intf:listContextResponse" name="listContextResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="ContextManagerSoapBinding" type="intf:ContextManagerImp">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="init">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="initRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
<wsdl:output name="initResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="removeContext">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="removeContextRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
<wsdl:output name="removeContextResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setContextStorage">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setContextStorageRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
<wsdl:output name="setContextStorageResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
```

```
  </wsdl:operation>
 <wsdl:operation name="addContext">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="addContextRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
 <wsdl:output name="addContextResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getCurrentProperty">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getCurrentPropertyRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getCurrentPropertyResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setCurrentProperty">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setCurrentPropertyRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setCurrentPropertyResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="listContext">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="listContextRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:input>
 <wsdl:output name="listContextResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="ContextManagerImpService">
 <wsdl:port binding="intf:ContextManagerSoapBinding" name="ContextManager">
  <wsdlsoap:address location="http://complexity.ucs.indiana.edu:8282/GCWS/services/ContextManager" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.5 Script Generator service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="urn:GatewayWS" xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="urn:GatewayWS" xmlns:intf="urn:GatewayWS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types />
 <wsdl:message name="writeStringToFileResponse">
  <wsdl:part name="return" type="xsd:boolean" />
```

```
  </wsdl:message>
<wsdl:message name="writeStringToFileRequest">
  <wsdl:part name="in0" type="soapenc:string" />
  <wsdl:part name="in1" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="scriptGenRequest">
  <wsdl:part name="in0" type="soapenc:string" />
  <wsdl:part name="in1" type="soapenc:string" />
  <wsdl:part name="in2" type="soapenc:string" />
  </wsdl:message>
<wsdl:message name="scriptGenResponse">
  <wsdl:part name="return" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="ScriptGeneratorImp">
<wsdl:operation name="scriptGen" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:scriptGenRequest" name="scriptGenRequest" />
  <wsdl:output message="intf:scriptGenResponse" name="scriptGenResponse" />
  </wsdl:operation>
<wsdl:operation name="writeStringToFile" parameterOrder="in0 in1">
  <wsdl:input message="intf:writeStringToFileRequest" name="writeStringToFileRequest" />
  <wsdl:output message="intf:writeStringToFileResponse" name="writeStringToFileResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="ScriptGeneratorSoapBinding" type="intf:ScriptGeneratorImp">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="scriptGen">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="scriptGenRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:GatewayWS" use="encoded" />
  </wsdl:input>
<wsdl:output name="scriptGenResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:GatewayWS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="writeStringToFile">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="writeStringToFileRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:GatewayWS" use="encoded" />
  </wsdl:input>
<wsdl:output name="writeStringToFileResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:GatewayWS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
<wsdl:service name="ScriptGeneratorImpService">
<wsdl:port binding="intf:ScriptGeneratorSoapBinding" name="ScriptGenerator">
  <wsdlsoap:address location="http://complexity.ucs.indiana.edu:8282/GCWS/services/ScriptGenerator" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.6 Application Descriptor service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2"
xmlns:intf="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
```

```
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="ArrayOf_soapenc_string">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOf_soapenc_string" nillable="true" type="tns1:ArrayOf_soapenc_string" />
  </schema>
  </wsdl:types>
<wsdl:message name="getErrorDescriptionResponse1">
  <wsdl:part name="getErrorDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputDescriptionResponse">
  <wsdl:part name="getOutputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorDescriptionRequest1">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getWalltimeDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputDescriptionRequest1">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="writeApplDescResponse" />
<wsdl:message name="getApplicationsResponse">
  <wsdl:part name="getApplicationsReturn" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
<wsdl:message name="setHostNameRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getErrorHandleResponse">
  <wsdl:part name="getErrorHandleReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="addApplHostRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  <wsdl:part name="in4" type="xsd:string" />
  <wsdl:part name="in5" type="xsd:string" />
  <wsdl:part name="in6" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getMemoryDirectiveResponse">
  <wsdl:part name="getMemoryDirectiveReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobBeginDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getNCPUDirectiveResponse">
  <wsdl:part name="getNCPUDirectiveReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostQueueTypeResponse">
  <wsdl:part name="getHostQueueTypeReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobAbortDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getMemoryDirectiveRequest">
```

```
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorMechResponse">
  <wsdl:part name="getErrorMechReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setQueueDirectivesRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  <wsdl:part name="in4" type="xsd:string" />
  <wsdl:part name="in5" type="xsd:string" />
  <wsdl:part name="in6" type="xsd:string" />
  <wsdl:part name="in7" type="xsd:string" />
  <wsdl:part name="in8" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostExecPathRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="setErrorDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getOutputDescriptionRequest1">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getWalltimeDirectiveResponse">
  <wsdl:part name="getWalltimeDirectiveReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getHostListNamesRequest" />
<wsdl:message name="setHostQueuePathRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="addApplicationRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in2" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in3" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in4" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in5" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in6" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in7" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in8" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in9" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
<wsdl:message name="setApplicationRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setErrorHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
```

```
<wsdl:message name="getHostListNamesResponse">
  <wsdl:part name="getHostListNamesReturn" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
  <wsdl:message name="setHostQueuePathResponse" />
<wsdl:message name="getInputDescriptionResponse1">
  <wsdl:part name="getInputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getApplicationsRequest" />
  <wsdl:message name="setInputHandleResponse" />
<wsdl:message name="getErrorPortCountResponse">
  <wsdl:part name="getErrorPortCountReturn" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="getOutputPortCountRequest" />
  <wsdl:message name="removeApplHostResponse" />
  <wsdl:message name="setHostExecPathResponse" />
  <wsdl:message name="setOutputMechResponse" />
<wsdl:message name="getOutputDescriptionResponse1">
  <wsdl:part name="getOutputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getQueueTypeResponse">
  <wsdl:part name="getQueueTypeReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostQueuePathRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setQueueDirectivesResponse" />
<wsdl:message name="getErrorHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getInputMechResponse">
  <wsdl:part name="getInputMechReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getErrorPortCountRequest" />
<wsdl:message name="getOutputHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getHostWorkDirResponse">
  <wsdl:part name="getHostWorkDirReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getQueueTypeRequest" />
<wsdl:message name="getErrorDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getHostQueueTypeRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputHandleResponse">
  <wsdl:part name="getOutputHandleReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobBeginDirectiveResponse">
  <wsdl:part name="getJobBeginDirectiveReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setHostQueueTypeRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setApplicationResponse" />
  <wsdl:message name="setErrorHandleResponse" />
  <wsdl:message name="setOutputHandleResponse" />
<wsdl:message name="writeApplDescRequest">
```

```
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostExecPathResponse">
  <wsdl:part name="getHostExecPathReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputMechResponse">
  <wsdl:part name="getOutputMechReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostWorkDirRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setHostIPRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setHostWorkDirRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobAbortDirectiveResponse">
  <wsdl:part name="getJobAbortDirectiveReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setErrorMechResponse" />
  <wsdl:message name="setHostQueueTypeResponse" />
<wsdl:message name="removeApplicationRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setHostExecPathRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setOutputMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="setOutputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getInputPortCountRequest" />
  <wsdl:message name="addApplHostResponse" />
  <wsdl:message name="addApplicationResponse" />
<wsdl:message name="readApplDescRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setOutputDescriptionResponse" />
<wsdl:message name="getHostIPResponse">
  <wsdl:part name="getHostIPReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="setErrorMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
```

```
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setInputMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorDescriptionResponse">
  <wsdl:part name="getErrorDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputDescriptionResponse">
  <wsdl:part name="getInputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setHostNameResponse" />
  <wsdl:message name="setErrorDescriptionResponse" />
<wsdl:message name="getJobEndDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setHostDescRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setInputDescriptionResponse" />
  <wsdl:message name="setHostIPResponse" />
  <wsdl:message name="readApplDescResponse" />
<wsdl:message name="getNCPUDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobEndDirectiveResponse">
  <wsdl:part name="getJobEndDirectiveReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setHostDescResponse" />
<wsdl:message name="getInputHandleResponse">
  <wsdl:part name="getInputHandleReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputPortCountResponse">
  <wsdl:part name="getInputPortCountReturn" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getOutputPortCountResponse">
  <wsdl:part name="getOutputPortCountReturn" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="removeApplHostRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostQueuePathResponse">
  <wsdl:part name="getHostQueuePathReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setInputMechResponse" />
<wsdl:message name="setOutputHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="removeApplicationResponse" />
<wsdl:message name="getHostIPRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getJobNameDirectiveRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
```

```
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="setHostWorkDirResponse" />
<wsdl:message name="getJobNameDirectiveResponse">
  <wsdl:part name="getJobNameDirectiveReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputMechRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="setInputHandleRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setInputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:int" />
  <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="ApplDescImpl2">
<wsdl:operation name="setHostName" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostNameRequest" name="setHostNameRequest" />
  <wsdl:output message="intf:setHostNameResponse" name="setHostNameResponse" />
  </wsdl:operation>
<wsdl:operation name="setApplication" parameterOrder="in0">
  <wsdl:input message="intf:setApplicationRequest" name="setApplicationRequest" />
  <wsdl:output message="intf:setApplicationResponse" name="setApplicationResponse" />
  </wsdl:operation>
<wsdl:operation name="readApplDesc" parameterOrder="in0">
  <wsdl:input message="intf:readApplDescRequest" name="readApplDescRequest" />
  <wsdl:output message="intf:readApplDescResponse" name="readApplDescResponse" />
  </wsdl:operation>
<wsdl:operation name="setHostDesc" parameterOrder="in0">
  <wsdl:input message="intf:setHostDescRequest" name="setHostDescRequest" />
  <wsdl:output message="intf:setHostDescResponse" name="setHostDescResponse" />
  </wsdl:operation>
<wsdl:operation name="getApplications">
  <wsdl:input message="intf:getApplicationsRequest" name="getApplicationsRequest" />
  <wsdl:output message="intf:getApplicationsResponse" name="getApplicationsResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputPortCount">
  <wsdl:input message="intf:getInputPortCountRequest" name="getInputPortCountRequest" />
  <wsdl:output message="intf:getInputPortCountResponse" name="getInputPortCountResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputDescription" parameterOrder="in0 in1">
  <wsdl:input message="intf:getInputDescriptionRequest" name="getInputDescriptionRequest" />
  <wsdl:output message="intf:getInputDescriptionResponse" name="getInputDescriptionResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputDescription" parameterOrder="in0">
  <wsdl:input message="intf:getInputDescriptionRequest1" name="getInputDescriptionRequest1" />
  <wsdl:output message="intf:getInputDescriptionResponse1" name="getInputDescriptionResponse1" />
  </wsdl:operation>
<wsdl:operation name="getOutputPortCount">
  <wsdl:input message="intf:getOutputPortCountRequest" name="getOutputPortCountRequest" />
  <wsdl:output message="intf:getOutputPortCountResponse" name="getOutputPortCountResponse" />
  </wsdl:operation>
<wsdl:operation name="getOutputDescription" parameterOrder="in0 in1">
  <wsdl:input message="intf:getOutputDescriptionRequest" name="getOutputDescriptionRequest" />
  <wsdl:output message="intf:getOutputDescriptionResponse" name="getOutputDescriptionResponse" />
  </wsdl:operation>
<wsdl:operation name="getOutputDescription" parameterOrder="in0">
  <wsdl:input message="intf:getOutputDescriptionRequest1" name="getOutputDescriptionRequest1" />
  <wsdl:output message="intf:getOutputDescriptionResponse1" name="getOutputDescriptionResponse1" />
  </wsdl:operation>
<wsdl:operation name="getErrorPortCount">
```

```
  <wsdl:input message="intf:getErrorPortCountRequest" name="getErrorPortCountRequest" />
  <wsdl:output message="intf:getErrorPortCountResponse" name="getErrorPortCountResponse" />
  </wsdl:operation>
 <wsdl:operation name="getErrorDescription" parameterOrder="in0 in1">
  <wsdl:input message="intf:getErrorDescriptionRequest" name="getErrorDescriptionRequest" />
  <wsdl:output message="intf:getErrorDescriptionResponse" name="getErrorDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getErrorDescription" parameterOrder="in0">
  <wsdl:input message="intf:getErrorDescriptionRequest1" name="getErrorDescriptionRequest1" />
  <wsdl:output message="intf:getErrorDescriptionResponse1" name="getErrorDescriptionResponse1" />
  </wsdl:operation>
 <wsdl:operation name="getHostListNames">
  <wsdl:input message="intf:getHostListNamesRequest" name="getHostListNamesRequest" />
  <wsdl:output message="intf:getHostListNamesResponse" name="getHostListNamesResponse" />
  </wsdl:operation>
 <wsdl:operation name="getQueueType">
  <wsdl:input message="intf:getQueueTypeRequest" name="getQueueTypeRequest" />
  <wsdl:output message="intf:getQueueTypeResponse" name="getQueueTypeResponse" />
  </wsdl:operation>
 <wsdl:operation name="getHostIP" parameterOrder="in0 in1">
  <wsdl:input message="intf:getHostIPRequest" name="getHostIPRequest" />
  <wsdl:output message="intf:getHostIPResponse" name="getHostIPResponse" />
  </wsdl:operation>
 <wsdl:operation name="setHostIP" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostIPRequest" name="setHostIPRequest" />
  <wsdl:output message="intf:setHostIPResponse" name="setHostIPResponse" />
  </wsdl:operation>
 <wsdl:operation name="getInputHandle" parameterOrder="in0 in1">
  <wsdl:input message="intf:getInputHandleRequest" name="getInputHandleRequest" />
  <wsdl:output message="intf:getInputHandleResponse" name="getInputHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setInputHandle" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setInputHandleRequest" name="setInputHandleRequest" />
  <wsdl:output message="intf:setInputHandleResponse" name="setInputHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setInputDescription" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setInputDescriptionRequest" name="setInputDescriptionRequest" />
  <wsdl:output message="intf:setInputDescriptionResponse" name="setInputDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getOutputHandle" parameterOrder="in0 in1">
  <wsdl:input message="intf:getOutputHandleRequest" name="getOutputHandleRequest" />
  <wsdl:output message="intf:getOutputHandleResponse" name="getOutputHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setOutputHandle" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setOutputHandleRequest" name="setOutputHandleRequest" />
  <wsdl:output message="intf:setOutputHandleResponse" name="setOutputHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setOutputDescription" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setOutputDescriptionRequest" name="setOutputDescriptionRequest" />
  <wsdl:output message="intf:setOutputDescriptionResponse" name="setOutputDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getErrorHandle" parameterOrder="in0 in1">
  <wsdl:input message="intf:getErrorHandleRequest" name="getErrorHandleRequest" />
  <wsdl:output message="intf:getErrorHandleResponse" name="getErrorHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setErrorHandle" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setErrorHandleRequest" name="setErrorHandleRequest" />
  <wsdl:output message="intf:setErrorHandleResponse" name="setErrorHandleResponse" />
  </wsdl:operation>
 <wsdl:operation name="setErrorDescription" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setErrorDescriptionRequest" name="setErrorDescriptionRequest" />
  <wsdl:output message="intf:setErrorDescriptionResponse" name="setErrorDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="writeApplDesc" parameterOrder="in0">
  <wsdl:input message="intf:writeApplDescRequest" name="writeApplDescRequest" />
  <wsdl:output message="intf:writeApplDescResponse" name="writeApplDescResponse" />
```

```
  </wsdl:operation>
<wsdl:operation name="setHostWorkDir" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostWorkDirRequest" name="setHostWorkDirRequest" />
  <wsdl:output message="intf:setHostWorkDirResponse" name="setHostWorkDirResponse" />
  </wsdl:operation>
<wsdl:operation name="setHostQueueType" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostQueueTypeRequest" name="setHostQueueTypeRequest" />
  <wsdl:output message="intf:setHostQueueTypeResponse" name="setHostQueueTypeResponse" />
  </wsdl:operation>
<wsdl:operation name="setHostExecPath" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostExecPathRequest" name="setHostExecPathRequest" />
  <wsdl:output message="intf:setHostExecPathResponse" name="setHostExecPathResponse" />
  </wsdl:operation>
<wsdl:operation name="setHostQueuePath" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setHostQueuePathRequest" name="setHostQueuePathRequest" />
  <wsdl:output message="intf:setHostQueuePathResponse" name="setHostQueuePathResponse" />
  </wsdl:operation>
<wsdl:operation name="getHostWorkDir" parameterOrder="in0 in1">
  <wsdl:input message="intf:getHostWorkDirRequest" name="getHostWorkDirRequest" />
  <wsdl:output message="intf:getHostWorkDirResponse" name="getHostWorkDirResponse" />
  </wsdl:operation>
<wsdl:operation name="getHostQueueType" parameterOrder="in0 in1">
  <wsdl:input message="intf:getHostQueueTypeRequest" name="getHostQueueTypeRequest" />
  <wsdl:output message="intf:getHostQueueTypeResponse" name="getHostQueueTypeResponse" />
  </wsdl:operation>
<wsdl:operation name="getHostExecPath" parameterOrder="in0 in1">
  <wsdl:input message="intf:getHostExecPathRequest" name="getHostExecPathRequest" />
  <wsdl:output message="intf:getHostExecPathResponse" name="getHostExecPathResponse" />
  </wsdl:operation>
<wsdl:operation name="getHostQueuePath" parameterOrder="in0 in1">
  <wsdl:input message="intf:getHostQueuePathRequest" name="getHostQueuePathRequest" />
  <wsdl:output message="intf:getHostQueuePathResponse" name="getHostQueuePathResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputMech" parameterOrder="in0 in1">
  <wsdl:input message="intf:getInputMechRequest" name="getInputMechRequest" />
  <wsdl:output message="intf:getInputMechResponse" name="getInputMechResponse" />
  </wsdl:operation>
<wsdl:operation name="getOutputMech" parameterOrder="in0 in1">
  <wsdl:input message="intf:getOutputMechRequest" name="getOutputMechRequest" />
  <wsdl:output message="intf:getOutputMechResponse" name="getOutputMechResponse" />
  </wsdl:operation>
<wsdl:operation name="getErrorMech" parameterOrder="in0 in1">
  <wsdl:input message="intf:getErrorMechRequest" name="getErrorMechRequest" />
  <wsdl:output message="intf:getErrorMechResponse" name="getErrorMechResponse" />
  </wsdl:operation>
<wsdl:operation name="setInputMech" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setInputMechRequest" name="setInputMechRequest" />
  <wsdl:output message="intf:setInputMechResponse" name="setInputMechResponse" />
  </wsdl:operation>
<wsdl:operation name="setOutputMech" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setOutputMechRequest" name="setOutputMechRequest" />
  <wsdl:output message="intf:setOutputMechResponse" name="setOutputMechResponse" />
  </wsdl:operation>
<wsdl:operation name="setErrorMech" parameterOrder="in0 in1 in2">
  <wsdl:input message="intf:setErrorMechRequest" name="setErrorMechRequest" />
  <wsdl:output message="intf:setErrorMechResponse" name="setErrorMechResponse" />
  </wsdl:operation>
<wsdl:operation name="addApplication" parameterOrder="in0 in1 in2 in3 in4 in5 in6 in7 in8 in9">
  <wsdl:input message="intf:addApplicationRequest" name="addApplicationRequest" />
  <wsdl:output message="intf:addApplicationResponse" name="addApplicationResponse" />
  </wsdl:operation>
<wsdl:operation name="addApplHost" parameterOrder="in0 in1 in2 in3 in4 in5 in6">
  <wsdl:input message="intf:addApplHostRequest" name="addApplHostRequest" />
  <wsdl:output message="intf:addApplHostResponse" name="addApplHostResponse" />
  </wsdl:operation>
<wsdl:operation name="removeApplication" parameterOrder="in0">
```

```
  <wsdl:input message="intf:removeApplicationRequest" name="removeApplicationRequest" />
  <wsdl:output message="intf:removeApplicationResponse" name="removeApplicationResponse" />
  </wsdl:operation>
 <wsdl:operation name="removeApplHost" parameterOrder="in0 in1">
  <wsdl:input message="intf:removeApplHostRequest" name="removeApplHostRequest" />
  <wsdl:output message="intf:removeApplHostResponse" name="removeApplHostResponse" />
  </wsdl:operation>
 <wsdl:operation name="setQueueDirectives" parameterOrder="in0 in1 in2 in3 in4 in5 in6 in7 in8">
  <wsdl:input message="intf:setQueueDirectivesRequest" name="setQueueDirectivesRequest" />
  <wsdl:output message="intf:setQueueDirectivesResponse" name="setQueueDirectivesResponse" />
  </wsdl:operation>
 <wsdl:operation name="getMemoryDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getMemoryDirectiveRequest" name="getMemoryDirectiveRequest" />
  <wsdl:output message="intf:getMemoryDirectiveResponse" name="getMemoryDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getJobNameDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getJobNameDirectiveRequest" name="getJobNameDirectiveRequest" />
  <wsdl:output message="intf:getJobNameDirectiveResponse" name="getJobNameDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getNCPUDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getNCPUDirectiveRequest" name="getNCPUDirectiveRequest" />
  <wsdl:output message="intf:getNCPUDirectiveResponse" name="getNCPUDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getWalltimeDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getWalltimeDirectiveRequest" name="getWalltimeDirectiveRequest" />
  <wsdl:output message="intf:getWalltimeDirectiveResponse" name="getWalltimeDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getJobBeginDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getJobBeginDirectiveRequest" name="getJobBeginDirectiveRequest" />
  <wsdl:output message="intf:getJobBeginDirectiveResponse" name="getJobBeginDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getJobEndDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getJobEndDirectiveRequest" name="getJobEndDirectiveRequest" />
  <wsdl:output message="intf:getJobEndDirectiveResponse" name="getJobEndDirectiveResponse" />
  </wsdl:operation>
 <wsdl:operation name="getJobAbortDirective" parameterOrder="in0 in1">
  <wsdl:input message="intf:getJobAbortDirectiveRequest" name="getJobAbortDirectiveRequest" />
  <wsdl:output message="intf:getJobAbortDirectiveResponse" name="getJobAbortDirectiveResponse" />
  </wsdl:operation>
  </wsdl:portType>
 <wsdl:binding name="ApplicationDescriptor2SoapBinding" type="intf:ApplDescImpl2">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="setHostName">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setHostNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setHostNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setApplication">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setApplicationRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setApplicationResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="readApplDesc">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="readApplDescRequest">
```

```
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="readApplDescResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setHostDesc">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setHostDescRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setHostDescResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getApplications">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getApplicationsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getApplicationsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputPortCount">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputPortCountRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputPortCountResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputDescriptionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputDescriptionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputDescriptionRequest1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputDescriptionResponse1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getOutputPortCount">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getOutputPortCountRequest">
```

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputPortCountResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getOutputDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getOutputDescriptionRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputDescriptionResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getOutputDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getOutputDescriptionRequest1">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputDescriptionResponse1">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorPortCount">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getErrorPortCountRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getErrorPortCountResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getErrorDescriptionRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getErrorDescriptionResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getErrorDescriptionRequest1">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getErrorDescriptionResponse1">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getHostListNames">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostListNamesRequest">

```
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getHostListNamesResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getQueueType">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getQueueTypeRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getQueueTypeResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getHostIP">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getHostIPRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getHostIPResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setHostIP">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setHostIPRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setHostIPResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputHandle">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputHandleRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputHandleResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setInputHandle">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setInputHandleRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setInputHandleResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setInputDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setInputDescriptionRequest">
```

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setInputDescriptionResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getOutputHandle">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getOutputHandleRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputHandleResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setOutputHandle">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setOutputHandleRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setOutputHandleResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setOutputDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setOutputDescriptionRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setOutputDescriptionResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorHandle">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getErrorHandleRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getErrorHandleResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setErrorHandle">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setErrorHandleRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setErrorHandleResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setErrorDescription">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setErrorDescriptionRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setErrorDescriptionResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="writeApplDesc">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="writeApplDescRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="writeApplDescResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setHostWorkDir">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setHostWorkDirRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setHostWorkDirResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setHostQueueType">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setHostQueueTypeRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setHostQueueTypeResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setHostExecPath">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setHostExecPathRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setHostExecPathResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setHostQueuePath">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setHostQueuePathRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setHostQueuePathResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getHostWorkDir">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostWorkDirRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getHostWorkDirResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getHostQueueType">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostQueueTypeRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getHostQueueTypeResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getHostExecPath">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostExecPathRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getHostExecPathResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getHostQueuePath">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostQueuePathRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getHostQueuePathResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getInputMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getInputMechRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getInputMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getOutputMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getOutputMechRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getErrorMechRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getErrorMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setInputMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setInputMechRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setInputMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setOutputMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setOutputMechRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setOutputMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setErrorMech">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setErrorMechRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setErrorMechResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addApplication">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="addApplicationRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="addApplicationResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="addApplHost">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="addApplHostRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="addApplHostResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="removeApplication">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="removeApplicationRequest">

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="removeApplicationResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="removeApplHost">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="removeApplHostRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="removeApplHostResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setQueueDirectives">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setQueueDirectivesRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="setQueueDirectivesResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getMemoryDirective">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getMemoryDirectiveRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getMemoryDirectiveResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getJobNameDirective">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getJobNameDirectiveRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getJobNameDirectiveResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getNCPUDirective">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getNCPUDirectiveRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:input>
<wsdl:output name="getNCPUDirectiveResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getWalltimeDirective">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getWalltimeDirectiveRequest">

```
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getWalltimeDirectiveResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getJobBeginDirective">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getJobBeginDirectiveRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getJobBeginDirectiveResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getJobEndDirective">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getJobEndDirectiveRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getJobEndDirectiveResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getJobAbortDirective">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getJobAbortDirectiveRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getJobAbortDirectiveResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
 <wsdl:service name="ApplDescImpl2Service">
 <wsdl:port binding="intf:ApplicationDescriptor2SoapBinding" name="ApplicationDescriptor2">
  <wsdlsoap:address location="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationDescriptor2" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.7 Application Instance service interface

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance"
xmlns:intf="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="ArrayOf_soapenc_string">
```

```
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOf_soapenc_string" nillable="true" type="tns1:ArrayOf_soapenc_string" />
  </schema>
  </wsdl:types>
  <wsdl:message name="getMemoryOptionRequest" />
<wsdl:message name="getOutputLocationRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getApplicationNameResponse">
  <wsdl:part name="getApplicationNameReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setInputPortRequest">
  <wsdl:part name="in0" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in1" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in2" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in3" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
<wsdl:message name="getInputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="setErrorPortRequest">
  <wsdl:part name="in0" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in1" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in2" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in3" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
<wsdl:message name="getInputDescriptionResponse">
  <wsdl:part name="getInputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getHostNameResponse">
  <wsdl:part name="getHostNameReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorDescriptionResponse">
  <wsdl:part name="getErrorDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getMemoryOptionResponse">
  <wsdl:part name="getMemoryOptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputLocationResponse">
  <wsdl:part name="getOutputLocationReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setWalltimeRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getOutputPortCountRequest" />
<wsdl:message name="getOutputPortCountResponse">
  <wsdl:part name="getOutputPortCountReturn" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="setMemoryOptionResponse" />
  <wsdl:message name="setEmailResponse" />
<wsdl:message name="getInputPortCountResponse">
  <wsdl:part name="getInputPortCountReturn" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getErrorPortCountResponse">
  <wsdl:part name="getErrorPortCountReturn" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="writeApplInsResponse" />
<wsdl:message name="createQueueInstanceRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
```

```
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  <wsdl:part name="in4" type="xsd:string" />
  <wsdl:part name="in5" type="xsd:string" />
  <wsdl:part name="in6" type="xsd:string" />
  <wsdl:part name="in7" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setOutputPortResponse" />
  <wsdl:message name="getHostNameRequest" />
  <wsdl:message name="setJobNameResponse" />
<wsdl:message name="getErrorLocationResponse">
  <wsdl:part name="getErrorLocationReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setJobNameRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getApplicationNameRequest" />
  <wsdl:message name="setInputPortResponse" />
  <wsdl:message name="createQueueInstanceResponse" />
  <wsdl:message name="getErrorPortCountRequest" />
<wsdl:message name="getJobNameResponse">
  <wsdl:part name="getJobNameReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setApplicationNameRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getQueueTypeResponse">
  <wsdl:part name="getQueueTypeReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setMemoryOptionRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getQueueTypeRequest" />
  <wsdl:message name="setApplicationNameResponse" />
<wsdl:message name="getApplInsStringResponse">
  <wsdl:part name="getApplInsStringReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setNumberOfCPUsRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getErrorDescriptionRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="createApplicationInstanceRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputDescriptionResponse">
  <wsdl:part name="getOutputDescriptionReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getInputLocationRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getErrorLocationRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="readApplInsResponse" />
  <wsdl:message name="getNumberOfCPUsRequest" />
<wsdl:message name="getWalltimeResponse">
  <wsdl:part name="getWalltimeReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getApplInsStringRequest" />
  <wsdl:message name="setWalltimeResponse" />
  <wsdl:message name="getJobNameRequest" />
```

```
<wsdl:message name="readApplInsRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getNumberOfCPUsResponse">
  <wsdl:part name="getNumberOfCPUsReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getWalltimeRequest" />
  <wsdl:message name="getQsubPathRequest" />
  <wsdl:message name="setErrorPortResponse" />
  <wsdl:message name="createApplicationInstanceResponse" />
<wsdl:message name="getInputLocationResponse">
  <wsdl:part name="getInputLocationReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getOutputDescriptionRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="writeApplInsRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getQsubPathResponse">
  <wsdl:part name="getQsubPathReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setOutputPortRequest">
  <wsdl:part name="in0" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in1" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in2" type="tns1:ArrayOf_soapenc_string" />
  <wsdl:part name="in3" type="tns1:ArrayOf_soapenc_string" />
  </wsdl:message>
  <wsdl:message name="setNumberOfCPUsResponse" />
<wsdl:message name="setEmailRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  <wsdl:part name="in4" type="xsd:string" />
  <wsdl:part name="in5" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getInputPortCountRequest" />
<wsdl:message name="createHostInstanceRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  <wsdl:part name="in2" type="xsd:string" />
  <wsdl:part name="in3" type="xsd:string" />
  <wsdl:part name="in4" type="xsd:string" />
  <wsdl:part name="in5" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="createHostInstanceResponse" />
<wsdl:portType name="ApplInstanceImpl">
<wsdl:operation name="getHostName">
  <wsdl:input message="intf:getHostNameRequest" name="getHostNameRequest" />
  <wsdl:output message="intf:getHostNameResponse" name="getHostNameResponse" />
  </wsdl:operation>
<wsdl:operation name="setEmail" parameterOrder="in0 in1 in2 in3 in4 in5">
  <wsdl:input message="intf:setEmailRequest" name="setEmailRequest" />
  <wsdl:output message="intf:setEmailResponse" name="setEmailResponse" />
  </wsdl:operation>
<wsdl:operation name="readApplIns" parameterOrder="in0">
  <wsdl:input message="intf:readApplInsRequest" name="readApplInsRequest" />
  <wsdl:output message="intf:readApplInsResponse" name="readApplInsResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputPortCount">
  <wsdl:input message="intf:getInputPortCountRequest" name="getInputPortCountRequest" />
  <wsdl:output message="intf:getInputPortCountResponse" name="getInputPortCountResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputDescription" parameterOrder="in0">
  <wsdl:input message="intf:getInputDescriptionRequest" name="getInputDescriptionRequest" />
```

```
  <wsdl:output message="intf:getInputDescriptionResponse" name="getInputDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getOutputPortCount">
  <wsdl:input message="intf:getOutputPortCountRequest" name="getOutputPortCountRequest" />
  <wsdl:output message="intf:getOutputPortCountResponse" name="getOutputPortCountResponse" />
  </wsdl:operation>
 <wsdl:operation name="getOutputDescription" parameterOrder="in0">
  <wsdl:input message="intf:getOutputDescriptionRequest" name="getOutputDescriptionRequest" />
  <wsdl:output message="intf:getOutputDescriptionResponse" name="getOutputDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getErrorPortCount">
  <wsdl:input message="intf:getErrorPortCountRequest" name="getErrorPortCountRequest" />
  <wsdl:output message="intf:getErrorPortCountResponse" name="getErrorPortCountResponse" />
  </wsdl:operation>
 <wsdl:operation name="getErrorDescription" parameterOrder="in0">
  <wsdl:input message="intf:getErrorDescriptionRequest" name="getErrorDescriptionRequest" />
  <wsdl:output message="intf:getErrorDescriptionResponse" name="getErrorDescriptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getQueueType">
  <wsdl:input message="intf:getQueueTypeRequest" name="getQueueTypeRequest" />
  <wsdl:output message="intf:getQueueTypeResponse" name="getQueueTypeResponse" />
  </wsdl:operation>
 <wsdl:operation name="getApplInsString">
  <wsdl:input message="intf:getApplInsStringRequest" name="getApplInsStringRequest" />
  <wsdl:output message="intf:getApplInsStringResponse" name="getApplInsStringResponse" />
  </wsdl:operation>
 <wsdl:operation name="getApplicationName">
  <wsdl:input message="intf:getApplicationNameRequest" name="getApplicationNameRequest" />
  <wsdl:output message="intf:getApplicationNameResponse" name="getApplicationNameResponse" />
  </wsdl:operation>
 <wsdl:operation name="setApplicationName" parameterOrder="in0">
  <wsdl:input message="intf:setApplicationNameRequest" name="setApplicationNameRequest" />
  <wsdl:output message="intf:setApplicationNameResponse" name="setApplicationNameResponse" />
  </wsdl:operation>
 <wsdl:operation name="getQsubPath">
  <wsdl:input message="intf:getQsubPathRequest" name="getQsubPathRequest" />
  <wsdl:output message="intf:getQsubPathResponse" name="getQsubPathResponse" />
  </wsdl:operation>
 <wsdl:operation name="setInputPort" parameterOrder="in0 in1 in2 in3">
  <wsdl:input message="intf:setInputPortRequest" name="setInputPortRequest" />
  <wsdl:output message="intf:setInputPortResponse" name="setInputPortResponse" />
  </wsdl:operation>
 <wsdl:operation name="setOutputPort" parameterOrder="in0 in1 in2 in3">
  <wsdl:input message="intf:setOutputPortRequest" name="setOutputPortRequest" />
  <wsdl:output message="intf:setOutputPortResponse" name="setOutputPortResponse" />
  </wsdl:operation>
 <wsdl:operation name="setErrorPort" parameterOrder="in0 in1 in2 in3">
  <wsdl:input message="intf:setErrorPortRequest" name="setErrorPortRequest" />
  <wsdl:output message="intf:setErrorPortResponse" name="setErrorPortResponse" />
  </wsdl:operation>
 <wsdl:operation name="setMemoryOption" parameterOrder="in0 in1">
  <wsdl:input message="intf:setMemoryOptionRequest" name="setMemoryOptionRequest" />
  <wsdl:output message="intf:setMemoryOptionResponse" name="setMemoryOptionResponse" />
  </wsdl:operation>
 <wsdl:operation name="getJobName">
  <wsdl:input message="intf:getJobNameRequest" name="getJobNameRequest" />
  <wsdl:output message="intf:getJobNameResponse" name="getJobNameResponse" />
  </wsdl:operation>
 <wsdl:operation name="setJobName" parameterOrder="in0 in1">
  <wsdl:input message="intf:setJobNameRequest" name="setJobNameRequest" />
  <wsdl:output message="intf:setJobNameResponse" name="setJobNameResponse" />
  </wsdl:operation>
 <wsdl:operation name="getNumberOfCPUs">
  <wsdl:input message="intf:getNumberOfCPUsRequest" name="getNumberOfCPUsRequest" />
  <wsdl:output message="intf:getNumberOfCPUsResponse" name="getNumberOfCPUsResponse" />
  </wsdl:operation>
```

```
<wsdl:operation name="setNumberOfCPUs" parameterOrder="in0 in1">
  <wsdl:input message="intf:setNumberOfCPUsRequest" name="setNumberOfCPUsRequest" />
  <wsdl:output message="intf:setNumberOfCPUsResponse" name="setNumberOfCPUsResponse" />
  </wsdl:operation>
<wsdl:operation name="getWalltime">
  <wsdl:input message="intf:getWalltimeRequest" name="getWalltimeRequest" />
  <wsdl:output message="intf:getWalltimeResponse" name="getWalltimeResponse" />
  </wsdl:operation>
<wsdl:operation name="setWalltime" parameterOrder="in0 in1">
  <wsdl:input message="intf:setWalltimeRequest" name="setWalltimeRequest" />
  <wsdl:output message="intf:setWalltimeResponse" name="setWalltimeResponse" />
  </wsdl:operation>
<wsdl:operation name="getInputLocation" parameterOrder="in0">
  <wsdl:input message="intf:getInputLocationRequest" name="getInputLocationRequest" />
  <wsdl:output message="intf:getInputLocationResponse" name="getInputLocationResponse" />
  </wsdl:operation>
<wsdl:operation name="getOutputLocation" parameterOrder="in0">
  <wsdl:input message="intf:getOutputLocationRequest" name="getOutputLocationRequest" />
  <wsdl:output message="intf:getOutputLocationResponse" name="getOutputLocationResponse" />
  </wsdl:operation>
<wsdl:operation name="getErrorLocation" parameterOrder="in0">
  <wsdl:input message="intf:getErrorLocationRequest" name="getErrorLocationRequest" />
  <wsdl:output message="intf:getErrorLocationResponse" name="getErrorLocationResponse" />
  </wsdl:operation>
<wsdl:operation name="getMemoryOption">
  <wsdl:input message="intf:getMemoryOptionRequest" name="getMemoryOptionRequest" />
  <wsdl:output message="intf:getMemoryOptionResponse" name="getMemoryOptionResponse" />
  </wsdl:operation>
<wsdl:operation name="createQueueInstance" parameterOrder="in0 in1 in2 in3 in4 in5 in6 in7">
  <wsdl:input message="intf:createQueueInstanceRequest" name="createQueueInstanceRequest" />
  <wsdl:output message="intf:createQueueInstanceResponse" name="createQueueInstanceResponse" />
  </wsdl:operation>
<wsdl:operation name="createHostInstance" parameterOrder="in0 in1 in2 in3 in4 in5">
  <wsdl:input message="intf:createHostInstanceRequest" name="createHostInstanceRequest" />
  <wsdl:output message="intf:createHostInstanceResponse" name="createHostInstanceResponse" />
  </wsdl:operation>
<wsdl:operation name="createApplicationInstance" parameterOrder="in0">
  <wsdl:input message="intf:createApplicationInstanceRequest" name="createApplicationInstanceRequest" />
  <wsdl:output message="intf:createApplicationInstanceResponse" name="createApplicationInstanceResponse" />
  </wsdl:operation>
<wsdl:operation name="writeApplIns" parameterOrder="in0">
  <wsdl:input message="intf:writeApplInsRequest" name="writeApplInsRequest" />
  <wsdl:output message="intf:writeApplInsResponse" name="writeApplInsResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="ApplicationInstanceSoapBinding" type="intf:ApplInstanceImpl">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getHostName">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getHostNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="getHostNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setEmail">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setEmailRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setEmailResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
```

```
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="readApplIns">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="readApplInsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="readApplInsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputPortCount">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputPortCountRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputPortCountResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getInputDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getInputDescriptionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getInputDescriptionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getOutputPortCount">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getOutputPortCountRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getOutputPortCountResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getOutputDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getOutputDescriptionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getOutputDescriptionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getErrorPortCount">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getErrorPortCountRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getErrorPortCountResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
```

```
  </wsdl:operation>
 <wsdl:operation name="getErrorDescription">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getErrorDescriptionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getErrorDescriptionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getQueueType">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getQueueTypeRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getQueueTypeResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getApplInsString">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getApplInsStringRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getApplInsStringResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getApplicationName">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getApplicationNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getApplicationNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setApplicationName">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setApplicationNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setApplicationNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getQsubPath">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getQsubPathRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getQsubPathResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
```

```
<wsdl:operation name="setInputPort">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setInputPortRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setInputPortResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setOutputPort">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setOutputPortRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setOutputPortResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setErrorPort">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setErrorPortRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setErrorPortResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setMemoryOption">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setMemoryOptionRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setMemoryOptionResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getJobName">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getJobNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="getJobNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setJobName">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setJobNameRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:input>
<wsdl:output name="setJobNameResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getNumberOfCPUs">
```

<wsdlsoap:operation soapAction="" />
<wsdl:input name="getNumberOfCPUsRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="getNumberOfCPUsResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setNumberOfCPUs">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setNumberOfCPUsRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="setNumberOfCPUsResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getWalltime">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getWalltimeRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="getWalltimeResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setWalltime">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setWalltimeRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="setWalltimeResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getInputLocation">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getInputLocationRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="getInputLocationResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getOutputLocation">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getOutputLocationRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:input>
<wsdl:output name="getOutputLocationResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getErrorLocation">
<wsdlsoap:operation soapAction="" />

```
 <wsdl:input name="getErrorLocationRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="getErrorLocationResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
 <wsdl:operation name="getMemoryOption">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getMemoryOptionRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="getMemoryOptionResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
 <wsdl:operation name="createQueueInstance">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="createQueueInstanceRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="createQueueInstanceResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
 <wsdl:operation name="createHostInstance">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="createHostInstanceRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="createHostInstanceResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
 <wsdl:operation name="createApplicationInstance">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="createApplicationInstanceRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="createApplicationInstanceResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
 <wsdl:operation name="writeApplIns">
   <wsdlsoap:operation soapAction="" />
 <wsdl:input name="writeApplInsRequest">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:input>
 <wsdl:output name="writeApplInsResponse">
   <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" use="encoded" />
   </wsdl:output>
   </wsdl:operation>
   </wsdl:binding>
 <wsdl:service name="ApplInstanceImplService">
 <wsdl:port binding="intf:ApplicationInstanceSoapBinding" name="ApplicationInstance">
```

```
  <wsdlsoap:address location="http://complexity.ucs.indiana.edu:8282/GCWS/services/ApplicationInstance" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.8 Data interaction service interface (Disloc code)

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS"
xmlns:intf="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="FaultData">
 <sequence>
   <element name="depthconstrained" type="xsd:boolean" />
   <element name="deptherror" type="xsd:float" />
   <element name="depthvalue" type="xsd:float" />
   <element name="dipSlipOrRateconstrained" type="xsd:boolean" />
   <element name="dipSlipOrRateerror" type="xsd:float" />
   <element name="dipSlipOrRatevalue" type="xsd:float" />
   <element name="dipconstrained" type="xsd:boolean" />
   <element name="diperror" type="xsd:float" />
   <element name="dipvalue" type="xsd:float" />
   <element name="lambda" type="xsd:float" />
   <element name="latitude" type="xsd:float" />
   <element name="lengthconstrained" type="xsd:boolean" />
   <element name="lengtherror" type="xsd:float" />
   <element name="lengthvalue" type="xsd:float" />
   <element name="longitude" type="xsd:float" />
   <element name="mu" type="xsd:float" />
   <element name="strikeSlipOrRateconstrained" type="xsd:boolean" />
   <element name="strikeSlipOrRateerror" type="xsd:float" />
   <element name="strikeSlipOrRatevalue" type="xsd:float" />
   <element name="strikeangle" type="xsd:float" />
   <element name="tensileSlipOrRateconstrained" type="xsd:boolean" />
   <element name="tensileSlipOrRateerror" type="xsd:float" />
   <element name="tensileSlipOrRatevalue" type="xsd:float" />
   <element name="widthconstrained" type="xsd:boolean" />
   <element name="widtherror" type="xsd:float" />
   <element name="widthvalue" type="xsd:float" />
   <element name="xcoordconstrained" type="xsd:boolean" />
   <element name="xcoorderror" type="xsd:float" />
   <element name="xcoordvalue" type="xsd:float" />
   <element name="ycoordconstrained" type="xsd:boolean" />
   <element name="ycoorderror" type="xsd:float" />
   <element name="ycoordvalue" type="xsd:float" />
   </sequence>
   </complexType>
   <element name="FaultData" nillable="true" type="tns1:FaultData" />
 <complexType name="ArrayOfFaultData">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:FaultData[]" />
   </restriction>
   </complexContent>
   </complexType>
```

```
  <element name="ArrayOfFaultData" nillable="true" type="tns1:ArrayOfFaultData" />
<complexType name="DisplacementData">
<sequence>
  <element name="adjDispXcalc" type="xsd:float" />
  <element name="adjDispXresid" type="xsd:float" />
  <element name="adjDispYcalc" type="xsd:float" />
  <element name="adjDispYresid" type="xsd:float" />
  <element name="adjDispZcalc" type="xsd:float" />
  <element name="adjDispZresid" type="xsd:float" />
  <element name="azimuth" type="xsd:float" />
  <element name="dispXcomp" type="xsd:float" />
  <element name="dispXerror" type="xsd:float" />
  <element name="dispYcomp" type="xsd:float" />
  <element name="dispYerror" type="xsd:float" />
  <element name="dispZcomp" type="xsd:float" />
  <element name="dispZerror" type="xsd:float" />
  <element name="elevation" type="xsd:float" />
  <element name="initDispXcalc" type="xsd:float" />
  <element name="initDispXresid" type="xsd:float" />
  <element name="initDispYcalc" type="xsd:float" />
  <element name="initDispYresid" type="xsd:float" />
  <element name="initDispZcalc" type="xsd:float" />
  <element name="initDispZresid" type="xsd:float" />
  <element name="locXcomp" type="xsd:float" />
  <element name="locXerror" type="xsd:float" />
  <element name="locYcomp" type="xsd:float" />
  <element name="locYerror" type="xsd:float" />
  <element name="observType" type="xsd:int" />
  </sequence>
  </complexType>
<complexType name="ArrayOfDisplacementData">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:DisplacementData[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOfDisplacementData" nillable="true" type="tns1:ArrayOfDisplacementData" />
<complexType name="GridObsvPoints">
<sequence>
  <element name="xnsteps" type="xsd:int" />
  <element name="xstart" type="xsd:float" />
  <element name="xstepsize" type="xsd:float" />
  <element name="ynsteps" type="xsd:int" />
  <element name="ystart" type="xsd:float" />
  <element name="ystepsize" type="xsd:float" />
  </sequence>
  </complexType>
  <element name="GridObsvPoints" nillable="true" type="tns1:GridObsvPoints" />
<complexType name="FreeObsvPoints">
<sequence>
  <element name="xcoord" type="xsd:float" />
  <element name="ycoord" type="xsd:float" />
  </sequence>
  </complexType>
<complexType name="ArrayOfFreeObsvPoints">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:FreeObsvPoints[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOfFreeObsvPoints" nillable="true" type="tns1:ArrayOfFreeObsvPoints" />
  </schema>
  </wsdl:types>
<wsdl:message name="getFreeObsvPointsRequest">
```

```
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="getAllFaultsRequest" />
<wsdl:message name="importInputDataResponse">
  <wsdl:part name="importInputDataReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="importOutputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setAllFaultsRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfFaultData" />
  </wsdl:message>
<wsdl:message name="getFreeObsvPointsResponse">
  <wsdl:part name="getFreeObsvPointsReturn" type="tns1:ArrayOfFreeObsvPoints" />
  </wsdl:message>
  <wsdl:message name="setAllDislocInputDataResponse" />
  <wsdl:message name="setGridObsvPointsResponse" />
<wsdl:message name="getFaultRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="exportOutputDataResponse" />
  <wsdl:message name="readDataResponse" />
  <wsdl:message name="setFaultResponse" />
<wsdl:message name="readDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setFaultRequest">
  <wsdl:part name="in0" type="tns1:FaultData" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getFaultResponse">
  <wsdl:part name="getFaultReturn" type="tns1:FaultData" />
  </wsdl:message>
<wsdl:message name="writeDataResponse">
  <wsdl:part name="writeDataReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="writeDataRequest1">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setFreeObsvPointsRequest">
  <wsdl:part name="in0" type="xsd:int" />
  <wsdl:part name="in1" type="tns1:ArrayOfFreeObsvPoints" />
  </wsdl:message>
  <wsdl:message name="setObservationStyleResponse" />
<wsdl:message name="getGridObsvPointsResponse">
  <wsdl:part name="getGridObsvPointsReturn" type="tns1:GridObsvPoints" />
  </wsdl:message>
  <wsdl:message name="getObservationStyleRequest" />
<wsdl:message name="getAllDisplacementsResponse">
  <wsdl:part name="getAllDisplacementsReturn" type="tns1:ArrayOfDisplacementData" />
  </wsdl:message>
<wsdl:message name="setAllDisplacementsRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfDisplacementData" />
  </wsdl:message>
<wsdl:message name="exportOutputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="importOutputDataResponse">
  <wsdl:part name="importOutputDataReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setAllFaultsResponse" />
  <wsdl:message name="setAllDisplacementsResponse" />
  <wsdl:message name="getAllDisplacementsRequest" />
<wsdl:message name="getObservationStyleResponse">
  <wsdl:part name="getObservationStyleReturn" type="xsd:int" />
```

```
  </wsdl:message>
<wsdl:message name="getAllFaultsResponse">
  <wsdl:part name="getAllFaultsReturn" type="tns1:ArrayOfFaultData" />
  </wsdl:message>
<wsdl:message name="getGridObsvPointsRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="writeDataRequest" />
  <wsdl:message name="writeDataResponse1" />
<wsdl:message name="importInputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="exportInputDataResponse" />
<wsdl:message name="setGridObsvPointsRequest">
  <wsdl:part name="in0" type="xsd:int" />
  <wsdl:part name="in1" type="tns1:GridObsvPoints" />
  </wsdl:message>
<wsdl:message name="exportInputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setAllDislocInputDataRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfFaultData" />
  <wsdl:part name="in1" type="tns1:GridObsvPoints" />
  </wsdl:message>
  <wsdl:message name="setFreeObsvPointsResponse" />
<wsdl:message name="setObservationStyleRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:portType name="DislocDataImpl">
<wsdl:operation name="getFault" parameterOrder="in0">
  <wsdl:input message="intf:getFaultRequest" name="getFaultRequest" />
  <wsdl:output message="intf:getFaultResponse" name="getFaultResponse" />
  </wsdl:operation>
<wsdl:operation name="setFault" parameterOrder="in0 in1">
  <wsdl:input message="intf:setFaultRequest" name="setFaultRequest" />
  <wsdl:output message="intf:setFaultResponse" name="setFaultResponse" />
  </wsdl:operation>
<wsdl:operation name="importInputData" parameterOrder="in0">
  <wsdl:input message="intf:importInputDataRequest" name="importInputDataRequest" />
  <wsdl:output message="intf:importInputDataResponse" name="importInputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="exportInputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:exportInputDataRequest" name="exportInputDataRequest" />
  <wsdl:output message="intf:exportInputDataResponse" name="exportInputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="readData" parameterOrder="in0">
  <wsdl:input message="intf:readDataRequest" name="readDataRequest" />
  <wsdl:output message="intf:readDataResponse" name="readDataResponse" />
  </wsdl:operation>
<wsdl:operation name="writeData">
  <wsdl:input message="intf:writeDataRequest" name="writeDataRequest" />
  <wsdl:output message="intf:writeDataResponse" name="writeDataResponse" />
  </wsdl:operation>
<wsdl:operation name="writeData" parameterOrder="in0">
  <wsdl:input message="intf:writeDataRequest1" name="writeDataRequest1" />
  <wsdl:output message="intf:writeDataResponse1" name="writeDataResponse1" />
  </wsdl:operation>
<wsdl:operation name="importOutputData" parameterOrder="in0">
  <wsdl:input message="intf:importOutputDataRequest" name="importOutputDataRequest" />
  <wsdl:output message="intf:importOutputDataResponse" name="importOutputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="exportOutputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:exportOutputDataRequest" name="exportOutputDataRequest" />
  <wsdl:output message="intf:exportOutputDataResponse" name="exportOutputDataResponse" />
  </wsdl:operation>
```

```
<wsdl:operation name="getAllFaults">
  <wsdl:input message="intf:getAllFaultsRequest" name="getAllFaultsRequest" />
  <wsdl:output message="intf:getAllFaultsResponse" name="getAllFaultsResponse" />
  </wsdl:operation>
<wsdl:operation name="setAllFaults" parameterOrder="in0">
  <wsdl:input message="intf:setAllFaultsRequest" name="setAllFaultsRequest" />
  <wsdl:output message="intf:setAllFaultsResponse" name="setAllFaultsResponse" />
  </wsdl:operation>
<wsdl:operation name="getAllDisplacements">
  <wsdl:input message="intf:getAllDisplacementsRequest" name="getAllDisplacementsRequest" />
  <wsdl:output message="intf:getAllDisplacementsResponse" name="getAllDisplacementsResponse" />
  </wsdl:operation>
<wsdl:operation name="setAllDisplacements" parameterOrder="in0">
  <wsdl:input message="intf:setAllDisplacementsRequest" name="setAllDisplacementsRequest" />
  <wsdl:output message="intf:setAllDisplacementsResponse" name="setAllDisplacementsResponse" />
  </wsdl:operation>
<wsdl:operation name="setObservationStyle" parameterOrder="in0">
  <wsdl:input message="intf:setObservationStyleRequest" name="setObservationStyleRequest" />
  <wsdl:output message="intf:setObservationStyleResponse" name="setObservationStyleResponse" />
  </wsdl:operation>
<wsdl:operation name="getObservationStyle">
  <wsdl:input message="intf:getObservationStyleRequest" name="getObservationStyleRequest" />
  <wsdl:output message="intf:getObservationStyleResponse" name="getObservationStyleResponse" />
  </wsdl:operation>
<wsdl:operation name="setGridObsvPoints" parameterOrder="in0 in1">
  <wsdl:input message="intf:setGridObsvPointsRequest" name="setGridObsvPointsRequest" />
  <wsdl:output message="intf:setGridObsvPointsResponse" name="setGridObsvPointsResponse" />
  </wsdl:operation>
<wsdl:operation name="getGridObsvPoints" parameterOrder="in0">
  <wsdl:input message="intf:getGridObsvPointsRequest" name="getGridObsvPointsRequest" />
  <wsdl:output message="intf:getGridObsvPointsResponse" name="getGridObsvPointsResponse" />
  </wsdl:operation>
<wsdl:operation name="setFreeObsvPoints" parameterOrder="in0 in1">
  <wsdl:input message="intf:setFreeObsvPointsRequest" name="setFreeObsvPointsRequest" />
  <wsdl:output message="intf:setFreeObsvPointsResponse" name="setFreeObsvPointsResponse" />
  </wsdl:operation>
<wsdl:operation name="getFreeObsvPoints" parameterOrder="in0">
  <wsdl:input message="intf:getFreeObsvPointsRequest" name="getFreeObsvPointsRequest" />
  <wsdl:output message="intf:getFreeObsvPointsResponse" name="getFreeObsvPointsResponse" />
  </wsdl:operation>
<wsdl:operation name="setAllDislocInputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:setAllDislocInputDataRequest" name="setAllDislocInputDataRequest" />
  <wsdl:output message="intf:setAllDislocInputDataResponse" name="setAllDislocInputDataResponse" />
  </wsdl:operation>
  </wsdl:portType>
<wsdl:binding name="DislocDSSoapBinding" type="intf:DislocDataImpl">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="getFault">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getFaultRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="getFaultResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setFault">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setFaultRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="setFaultResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
```

```
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="importInputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="importInputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="importInputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="exportInputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exportInputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="exportInputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="readData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="readDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="readDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="writeData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="writeDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="writeDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="writeData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="writeDataRequest1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="writeDataResponse1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="importOutputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="importOutputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="importOutputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
```

```
  </wsdl:operation>
 <wsdl:operation name="exportOutputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exportOutputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="exportOutputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getAllFaults">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getAllFaultsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getAllFaultsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setAllFaults">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setAllFaultsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setAllFaultsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getAllDisplacements">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getAllDisplacementsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getAllDisplacementsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setAllDisplacements">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setAllDisplacementsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setAllDisplacementsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setObservationStyle">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setObservationStyleRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setObservationStyleResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
```

```
<wsdl:operation name="getObservationStyle">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getObservationStyleRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="getObservationStyleResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setGridObsvPoints">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setGridObsvPointsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="setGridObsvPointsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getGridObsvPoints">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getGridObsvPointsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="getGridObsvPointsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setFreeObsvPoints">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setFreeObsvPointsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="setFreeObsvPointsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="getFreeObsvPoints">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="getFreeObsvPointsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="getFreeObsvPointsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
<wsdl:operation name="setAllDislocInputData">
  <wsdlsoap:operation soapAction="" />
<wsdl:input name="setAllDislocInputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:input>
<wsdl:output name="setAllDislocInputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
```

```
<wsdl:service name="DislocDataImplService">
<wsdl:port binding="intf:DislocDSSoapBinding" name="DislocDS">
  <wsdlsoap:address location="http://solar.uits.indiana.edu:8005/GCWS/services/DislocDS" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```

A.9 Data interaction service interface (Simplex code)

```
<?xml version="1.0" encoding="UTF-8" ?>
 <wsdl:definitions targetNamespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS"
xmlns:intf="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns1="urn:GatewayWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema targetNamespace="urn:GatewayWS" xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
 <complexType name="FaultData">
 <sequence>
   <element name="depthconstrained" type="xsd:boolean" />
   <element name="deptherror" type="xsd:float" />
   <element name="depthvalue" type="xsd:float" />
   <element name="dipSlipOrRateconstrained" type="xsd:boolean" />
   <element name="dipSlipOrRateerror" type="xsd:float" />
   <element name="dipSlipOrRatevalue" type="xsd:float" />
   <element name="dipconstrained" type="xsd:boolean" />
   <element name="diperror" type="xsd:float" />
   <element name="dipvalue" type="xsd:float" />
   <element name="lambda" type="xsd:float" />
   <element name="latitude" type="xsd:float" />
   <element name="lengthconstrained" type="xsd:boolean" />
   <element name="lengtherror" type="xsd:float" />
   <element name="lengthvalue" type="xsd:float" />
   <element name="longitude" type="xsd:float" />
   <element name="mu" type="xsd:float" />
   <element name="strikeSlipOrRateconstrained" type="xsd:boolean" />
   <element name="strikeSlipOrRateerror" type="xsd:float" />
   <element name="strikeSlipOrRatevalue" type="xsd:float" />
   <element name="strikeangle" type="xsd:float" />
   <element name="tensileSlipOrRateconstrained" type="xsd:boolean" />
   <element name="tensileSlipOrRateerror" type="xsd:float" />
   <element name="tensileSlipOrRatevalue" type="xsd:float" />
   <element name="widthconstrained" type="xsd:boolean" />
   <element name="widtherror" type="xsd:float" />
   <element name="widthvalue" type="xsd:float" />
   <element name="xcoordconstrained" type="xsd:boolean" />
   <element name="xcoorderror" type="xsd:float" />
   <element name="xcoordvalue" type="xsd:float" />
   <element name="ycoordconstrained" type="xsd:boolean" />
   <element name="ycoorderror" type="xsd:float" />
   <element name="ycoordvalue" type="xsd:float" />
   </sequence>
   </complexType>
   <element name="FaultData" nillable="true" type="tns1:FaultData" />
 <complexType name="ArrayOfFaultData">
 <complexContent>
 <restriction base="soapenc:Array">
   <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:FaultData[]" />
   </restriction>
```

```
  </complexContent>
  </complexType>
  <element name="ArrayOfFaultData" nillable="true" type="tns1:ArrayOfFaultData" />
<complexType name="DisplacementData">
<sequence>
  <element name="adjDispXcalc" type="xsd:float" />
  <element name="adjDispXresid" type="xsd:float" />
  <element name="adjDispYcalc" type="xsd:float" />
  <element name="adjDispYresid" type="xsd:float" />
  <element name="adjDispZcalc" type="xsd:float" />
  <element name="adjDispZresid" type="xsd:float" />
  <element name="azimuth" type="xsd:float" />
  <element name="dispXcomp" type="xsd:float" />
  <element name="dispXerror" type="xsd:float" />
  <element name="dispYcomp" type="xsd:float" />
  <element name="dispYerror" type="xsd:float" />
  <element name="dispZcomp" type="xsd:float" />
  <element name="dispZerror" type="xsd:float" />
  <element name="elevation" type="xsd:float" />
  <element name="initDispXcalc" type="xsd:float" />
  <element name="initDispXresid" type="xsd:float" />
  <element name="initDispYcalc" type="xsd:float" />
  <element name="initDispYresid" type="xsd:float" />
  <element name="initDispZcalc" type="xsd:float" />
  <element name="initDispZresid" type="xsd:float" />
  <element name="locXcomp" type="xsd:float" />
  <element name="locXerror" type="xsd:float" />
  <element name="locYcomp" type="xsd:float" />
  <element name="locYerror" type="xsd:float" />
  <element name="observType" type="xsd:int" />
  </sequence>
  </complexType>
<complexType name="ArrayOfDisplacementData">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:DisplacementData[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOfDisplacementData" nillable="true" type="tns1:ArrayOfDisplacementData" />
<complexType name="FaultParameter">
<sequence>
  <element name="activeparam" type="xsd:int" />
  <element name="faultparamnum" type="xsd:int" />
  </sequence>
  </complexType>
<complexType name="ArrayOfFaultParameter">
<complexContent>
<restriction base="soapenc:Array">
  <attribute ref="soapenc:arrayType" wsdl:arrayType="tns1:FaultParameter[]" />
  </restriction>
  </complexContent>
  </complexType>
  <element name="ArrayOfFaultParameter" nillable="true" type="tns1:ArrayOfFaultParameter" />
<complexType name="CodeParameter">
<sequence>
  <element name="maxiter" type="xsd:int" />
  <element name="starttemp" type="xsd:float" />
  </sequence>
  </complexType>
  <element name="CodeParameter" nillable="true" type="tns1:CodeParameter" />
  </schema>
  </wsdl:types>
  <wsdl:message name="getAllFaultsRequest" />
<wsdl:message name="importInputDataResponse">
  <wsdl:part name="importInputDataReturn" type="xsd:string" />
```

```
  </wsdl:message>
  <wsdl:message name="setBestchi2Response" />
  <wsdl:message name="setCodeParameterResponse" />
<wsdl:message name="importOutputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setAllFaultsRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfFaultData" />
  </wsdl:message>
<wsdl:message name="getFaultRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="exportOutputDataResponse" />
  <wsdl:message name="readDataResponse" />
  <wsdl:message name="setFaultResponse" />
  <wsdl:message name="getBestchi2Request" />
  <wsdl:message name="setFaultParameterResponse" />
<wsdl:message name="writeDataResponse">
  <wsdl:part name="writeDataReturn" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="writeDataRequest1">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getBestchi2Response">
  <wsdl:part name="getBestchi2Return" type="xsd:double" />
  </wsdl:message>
  <wsdl:message name="setAllSimplexInputDataResponse" />
<wsdl:message name="readDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="setFaultRequest">
  <wsdl:part name="in0" type="tns1:FaultData" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="getFaultResponse">
  <wsdl:part name="getFaultReturn" type="tns1:FaultData" />
  </wsdl:message>
<wsdl:message name="getFaultParameterRequest">
  <wsdl:part name="in0" type="xsd:int" />
  </wsdl:message>
<wsdl:message name="getAllDisplacementsResponse">
  <wsdl:part name="getAllDisplacementsReturn" type="tns1:ArrayOfDisplacementData" />
  </wsdl:message>
<wsdl:message name="setAllDisplacementsRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfDisplacementData" />
  </wsdl:message>
<wsdl:message name="exportOutputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:message name="importOutputDataResponse">
  <wsdl:part name="importOutputDataReturn" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="setAllFaultsResponse" />
<wsdl:message name="setCodeParameterRequest">
  <wsdl:part name="in0" type="tns1:CodeParameter" />
  </wsdl:message>
<wsdl:message name="setBestchi2Request">
  <wsdl:part name="in0" type="xsd:double" />
  </wsdl:message>
<wsdl:message name="getFaultParameterResponse">
  <wsdl:part name="getFaultParameterReturn" type="tns1:ArrayOfFaultParameter" />
  </wsdl:message>
  <wsdl:message name="setAllDisplacementsResponse" />
  <wsdl:message name="getAllDisplacementsRequest" />
<wsdl:message name="getAllFaultsResponse">
```

```
  <wsdl:part name="getAllFaultsReturn" type="tns1:ArrayOfFaultData" />
  </wsdl:message>
<wsdl:message name="setAllSimplexInputDataRequest">
  <wsdl:part name="in0" type="tns1:ArrayOfFaultData" />
  <wsdl:part name="in1" type="tns1:ArrayOfDisplacementData" />
  </wsdl:message>
<wsdl:message name="importInputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="exportInputDataResponse" />
<wsdl:message name="setFaultParameterRequest">
  <wsdl:part name="in0" type="xsd:int" />
  <wsdl:part name="in1" type="tns1:ArrayOfFaultParameter" />
  </wsdl:message>
  <wsdl:message name="writeDataRequest" />
<wsdl:message name="getCodeParameterResponse">
  <wsdl:part name="getCodeParameterReturn" type="tns1:CodeParameter" />
  </wsdl:message>
  <wsdl:message name="writeDataResponse1" />
  <wsdl:message name="getCodeParameterRequest" />
<wsdl:message name="exportInputDataRequest">
  <wsdl:part name="in0" type="xsd:string" />
  <wsdl:part name="in1" type="xsd:string" />
  </wsdl:message>
<wsdl:portType name="SimplexDataImpl">
<wsdl:operation name="getFault" parameterOrder="in0">
  <wsdl:input message="intf:getFaultRequest" name="getFaultRequest" />
  <wsdl:output message="intf:getFaultResponse" name="getFaultResponse" />
  </wsdl:operation>
<wsdl:operation name="setFault" parameterOrder="in0 in1">
  <wsdl:input message="intf:setFaultRequest" name="setFaultRequest" />
  <wsdl:output message="intf:setFaultResponse" name="setFaultResponse" />
  </wsdl:operation>
<wsdl:operation name="importInputData" parameterOrder="in0">
  <wsdl:input message="intf:importInputDataRequest" name="importInputDataRequest" />
  <wsdl:output message="intf:importInputDataResponse" name="importInputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="exportInputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:exportInputDataRequest" name="exportInputDataRequest" />
  <wsdl:output message="intf:exportInputDataResponse" name="exportInputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="readData" parameterOrder="in0">
  <wsdl:input message="intf:readDataRequest" name="readDataRequest" />
  <wsdl:output message="intf:readDataResponse" name="readDataResponse" />
  </wsdl:operation>
<wsdl:operation name="writeData">
  <wsdl:input message="intf:writeDataRequest" name="writeDataRequest" />
  <wsdl:output message="intf:writeDataResponse" name="writeDataResponse" />
  </wsdl:operation>
<wsdl:operation name="writeData" parameterOrder="in0">
  <wsdl:input message="intf:writeDataRequest1" name="writeDataRequest1" />
  <wsdl:output message="intf:writeDataResponse1" name="writeDataResponse1" />
  </wsdl:operation>
<wsdl:operation name="importOutputData" parameterOrder="in0">
  <wsdl:input message="intf:importOutputDataRequest" name="importOutputDataRequest" />
  <wsdl:output message="intf:importOutputDataResponse" name="importOutputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="exportOutputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:exportOutputDataRequest" name="exportOutputDataRequest" />
  <wsdl:output message="intf:exportOutputDataResponse" name="exportOutputDataResponse" />
  </wsdl:operation>
<wsdl:operation name="getAllFaults">
  <wsdl:input message="intf:getAllFaultsRequest" name="getAllFaultsRequest" />
  <wsdl:output message="intf:getAllFaultsResponse" name="getAllFaultsResponse" />
  </wsdl:operation>
<wsdl:operation name="setAllFaults" parameterOrder="in0">
```

```
  <wsdl:input message="intf:setAllFaultsRequest" name="setAllFaultsRequest" />
  <wsdl:output message="intf:setAllFaultsResponse" name="setAllFaultsResponse" />
  </wsdl:operation>
 <wsdl:operation name="getAllDisplacements">
  <wsdl:input message="intf:getAllDisplacementsRequest" name="getAllDisplacementsRequest" />
  <wsdl:output message="intf:getAllDisplacementsResponse" name="getAllDisplacementsResponse" />
  </wsdl:operation>
 <wsdl:operation name="setAllDisplacements" parameterOrder="in0">
  <wsdl:input message="intf:setAllDisplacementsRequest" name="setAllDisplacementsRequest" />
  <wsdl:output message="intf:setAllDisplacementsResponse" name="setAllDisplacementsResponse" />
  </wsdl:operation>
 <wsdl:operation name="setFaultParameter" parameterOrder="in0 in1">
  <wsdl:input message="intf:setFaultParameterRequest" name="setFaultParameterRequest" />
  <wsdl:output message="intf:setFaultParameterResponse" name="setFaultParameterResponse" />
  </wsdl:operation>
 <wsdl:operation name="getFaultParameter" parameterOrder="in0">
  <wsdl:input message="intf:getFaultParameterRequest" name="getFaultParameterRequest" />
  <wsdl:output message="intf:getFaultParameterResponse" name="getFaultParameterResponse" />
  </wsdl:operation>
 <wsdl:operation name="setCodeParameter" parameterOrder="in0">
  <wsdl:input message="intf:setCodeParameterRequest" name="setCodeParameterRequest" />
  <wsdl:output message="intf:setCodeParameterResponse" name="setCodeParameterResponse" />
  </wsdl:operation>
 <wsdl:operation name="getCodeParameter">
  <wsdl:input message="intf:getCodeParameterRequest" name="getCodeParameterRequest" />
  <wsdl:output message="intf:getCodeParameterResponse" name="getCodeParameterResponse" />
  </wsdl:operation>
 <wsdl:operation name="setBestchi2" parameterOrder="in0">
  <wsdl:input message="intf:setBestchi2Request" name="setBestchi2Request" />
  <wsdl:output message="intf:setBestchi2Response" name="setBestchi2Response" />
  </wsdl:operation>
 <wsdl:operation name="getBestchi2">
  <wsdl:input message="intf:getBestchi2Request" name="getBestchi2Request" />
  <wsdl:output message="intf:getBestchi2Response" name="getBestchi2Response" />
  </wsdl:operation>
 <wsdl:operation name="setAllSimplexInputData" parameterOrder="in0 in1">
  <wsdl:input message="intf:setAllSimplexInputDataRequest" name="setAllSimplexInputDataRequest" />
  <wsdl:output message="intf:setAllSimplexInputDataResponse" name="setAllSimplexInputDataResponse" />
  </wsdl:operation>
  </wsdl:portType>
 <wsdl:binding name="SimplexDSSoapBinding" type="intf:SimplexDataImpl">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="getFault">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getFaultRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getFaultResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setFault">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setFaultRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setFaultResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="importInputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="importInputDataRequest">
```

```
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="importInputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="exportInputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exportInputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="exportInputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="readData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="readDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="readDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="writeData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="writeDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="writeDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="writeData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="writeDataRequest1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="writeDataResponse1">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="importOutputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="importOutputDataRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="importOutputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="exportOutputData">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="exportOutputDataRequest">
```

```
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="exportOutputDataResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getAllFaults">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getAllFaultsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getAllFaultsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setAllFaults">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setAllFaultsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setAllFaultsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getAllDisplacements">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getAllDisplacementsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="getAllDisplacementsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setAllDisplacements">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setAllDisplacementsRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setAllDisplacementsResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="setFaultParameter">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="setFaultParameterRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:input>
 <wsdl:output name="setFaultParameterResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
  </wsdl:output>
  </wsdl:operation>
 <wsdl:operation name="getFaultParameter">
  <wsdlsoap:operation soapAction="" />
 <wsdl:input name="getFaultParameterRequest">
```

<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="getFaultParameterResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setCodeParameter">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setCodeParameterRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="setCodeParameterResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getCodeParameter">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getCodeParameterRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="getCodeParameterResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setBestchi2">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setBestchi2Request">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="setBestchi2Response">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getBestchi2">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getBestchi2Request">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="getBestchi2Response">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setAllSimplexInputData">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setAllSimplexInputDataRequest">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:input>
<wsdl:output name="setAllSimplexInputDataResponse">
<wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" use="encoded" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="SimplexDataImplService">
<wsdl:port binding="intf:SimplexDSSoapBinding" name="SimplexDS">
<wsdlsoap:address location="http://solar.uits.indiana.edu:8005/GCWS/services/SimplexDS" />

```
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Appendix B

XML schema

B.1 Context Manager XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8045/Schema/CM" xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:cm="http://grids.ucs.indiana.edu:8045/Schema/CM"
elementFormDefault="qualified">
    <xsd:complexType name="contextdataType">
        <xsd:sequence>
            <xsd:element name="Name" type="string"/>
            <xsd:element name="Value" type="string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="contextType">
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element name="Context" type="cm:contextType"/>
            <xsd:element name="ContextData" type="cm:contextdataType"/>
            <xsd:element name="Name" type="string"/>
        </xsd:choice>
    </xsd:complexType>
    <xsd:element name="ContextManager">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Context" type="cm:contextType" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

B.2 Application Descriptor XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Appl2"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:appl="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Appl2">
    <xsd:annotation>
        <xsd:documentation>
     This schema is used to describe an application.
      </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="ApplDesc" type="appl:ApplDescType"/>
    <xsd:complexType name="ApplDescType">
        <xsd:sequence>
            <xsd:element name="Application" type="appl:ApplicationType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ApplicationType">
        <xsd:sequence>
            <xsd:element name="BaseInfo" type="appl:BaseInfoType"/>
            <xsd:element name="InternalComm" type="appl:InternalCommType"/>
            <xsd:element name="ExecBundle" type="appl:ExecBundleType"/>
            <xsd:element name="ExternalComm" type="appl:ExternalCommType"/>
            <xsd:element name="ApplicationParameter" type="appl:ParameterType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BaseInfoType">
        <xsd:sequence>
            <xsd:element name="ApplicationName" type="xsd:string"/>
            <xsd:element name="Version" type="xsd:string" minOccurs="0"/>
            <xsd:element name="OptionFlag" type="appl:OptionFlagType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="OptionFlagType">
        <xsd:sequence>
            <xsd:element name="flagName" type="xsd:string"/>
            <xsd:element name="flagValue" type="xsd:boolean" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="InternalCommType">
        <xsd:sequence>
            <xsd:element name="InputField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="OutputField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="ErrorField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FieldType">
        <xsd:sequence>
            <xsd:element name="Handle" type="xsd:string"/>
            <xsd:element name="Description" type="xsd:string"/>
            <xsd:element name="IOMechanism" type="appl:IOMechanismType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="IOMechanismType">
        <xsd:sequence>
            <xsd:element name="localMech" type="appl:simpleMechanism"/>
            <xsd:element name="remoteMech" type="appl:ServiceType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="simpleMechanism">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="StandardIO"/>
            <xsd:enumeration value="CArgument"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <!--Note for now the order implies the sequence -->
    <xsd:complexType name="ExecBundleType">
```

```
        <xsd:sequence>
            <xsd:element name="Service" type="appl:ServiceType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--ServiceDef must point to the WSDL definition of the -->
    <!--service.-->
    <xsd:complexType name="ServiceType">
        <xsd:sequence>
            <xsd:element name="ServiceName" type="xsd:string"/>
            <xsd:element name="ServiceDesc" type="xsd:string"/>
            <xsd:element name="ServiceDef" type="xsd:anyURI"/>
            <xsd:element name="HostBinding" type="appl:HostBindingType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--ServiceBindingPoint is the URI of the SOAP server -->
    <xsd:complexType name="HostBindingType">
        <xsd:sequence>
            <xsd:element name="ServiceBindingPoint" type="xsd:anyURI"/>
            <xsd:element name="HostDesc" type="xsd:anyType" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="HostName" type="xsd:string"/>
    </xsd:complexType>
    <!--This is empty for now but can later be filled with -->
    <!--ServiceType entries.-->
    <xsd:complexType name="ExternalCommType"/>
</xsd:schema>
```

B.3 Application Instance XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Applins2"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:appl="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Applins2">
    <xsd:annotation>
        <xsd:documentation>
This schema is used to describe an application instance.  It is intended
to work with Appl2.xsd, the revised Application Descriptor Schema.  The
intended difference is that the instance schema holds particular application
choices (hence only one Application element per doc).  The ApplDesc schema
is used to describe all possible choices.
</xsd:documentation>
    </xsd:annotation>
    <xsd:element name="ApplInstance" type="appl:ApplInstanceType"/>
    <xsd:complexType name="ApplInstanceType">
        <xsd:sequence>
            <xsd:element name="BaseInfo" type="appl:BaseInfoType"/>
            <xsd:element name="InternalComm" type="appl:InternalCommType"/>
            <xsd:element name="ExecBundle" type="appl:ExecBundleType"/>
            <xsd:element name="ExternalComm" type="appl:ExternalCommType"/>
            <xsd:element name="ApplicationParameter" type="appl:ParameterType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="BaseInfoType">
        <xsd:sequence>
            <xsd:element name="ApplicationName" type="xsd:string"/>
            <xsd:element name="Version" type="xsd:string" minOccurs="0"/>
            <xsd:element name="OptionFlag" type="appl:OptionFlagType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="OptionFlagType">
        <xsd:sequence>
            <xsd:element name="flagName" type="xsd:string"/>
            <xsd:element name="flagValue" type="xsd:boolean" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--
```

```
InternalComms are used to describe how the application handles communication
in its internal parts.
-->
    <xsd:complexType name="InternalCommType">
        <xsd:sequence>
            <xsd:element name="InputField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="OutputField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="ErrorField" type="appl:FieldType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--
Field elements:
* Handle is a useful internal reference name.
* Description is a longer description of the field.
* IOMechanism is how the code interacts with its environment (i.e. how
  does it get input and write output?)
-->
    <xsd:complexType name="FieldType">
        <xsd:sequence>
            <xsd:element name="Handle" type="xsd:string"/>
            <xsd:element name="Description" type="xsd:string"/>
            <xsd:element name="Location" type="xsd:string"/>
            <xsd:element name="IOMechanism" type="appl:IOMechanismType"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--
All codes are presumed to require one local I or O mechanism per field.
Optionally codes have these IO elements (particularly output) redirected to
numerous places.  For example, output from a code run may be stored
to disk, emailed to me, and uploaded to a remote file system.  The
remoteMech is just a service.
-->
    <xsd:complexType name="IOMechanismType">
        <xsd:sequence>
            <xsd:element name="localMech" type="appl:simpleMechanism"/>
            <xsd:element name="remoteMech" type="appl:ServiceType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="simpleMechanism">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="StandardIO"/>
            <xsd:enumeration value="CArgument"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <!--Note for now the order implies the sequence -->
    <xsd:complexType name="ExecBundleType">
        <xsd:sequence>
            <xsd:element name="Service" type="appl:ServiceType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--Services are web services.  Must point to the WSDL definition of the -->
    <!--service.-->
    <xsd:complexType name="ServiceType">
        <xsd:sequence>
            <xsd:element name="ServiceName" type="xsd:string"/>
            <xsd:element name="ServiceDesc" type="xsd:string"/>
            <xsd:element name="ServiceDef" type="xsd:anyURI"/>
            <xsd:element name="HostBindingInstance" type="appl:HostBindingType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--ServiceBindingPoint is the URI of the SOAP server -->
    <xsd:complexType name="HostBindingType">
        <xsd:sequence>
            <xsd:element name="ServiceBindingPoint" type="xsd:anyURI"/>
```

```
            <xsd:element name="HostDescIns" type="xsd:anyType" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="HostName" type="xsd:string"/>
    </xsd:complexType>
    <!--This is empty for now but can later be filled with -->
    <!--ServiceType entries.-->
    <xsd:complexType name="ExternalCommType"/>
</xsd:schema>
```

B.4 Host Descriptor XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:host="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host">
    <xsd:annotation>
        <xsd:documentation>
      This schema describes host machine bindings for applications.
       </xsd:documentation>
    </xsd:annotation>
    <!--
  HostDesc is the root element, with any number of applications.
  HostDesc contains a description of a host machine. It contains
  several required fields and can be extended by an arbitrary
  number of parameters.  Optionally, you can provide commands
  for remote copy and execution (rsh, rcp, globusrun).  This must
  be set up externally.

  Hosts also contain one or more bindings for queue execution.  This
  could be a queuing system (PBS, GRD) or it could be
  -->
    <xsd:element name="HostDesc" type="host:HostDescType"/>
    <xsd:complexType name="HostDescType">
        <xsd:sequence>
            <xsd:element name="HostName" type="xsd:string"/>
            <xsd:element name="HostIP" type="xsd:string"/>
            <xsd:element name="WorkDir" type="xsd:string"/>
            <xsd:element name="QueueType" type="xsd:string"/>
            <xsd:element name="QsubPath" type="xsd:string"/>
            <xsd:element name="ExecPath" type="xsd:string"/>
            <xsd:element name="RemoteCopy" type="xsd:string" minOccurs="0"/>
            <xsd:element name="RemoteExec" type="xsd:string" minOccurs="0"/>
            <xsd:element name="HostParameter" type="host:ParameterType" minOccurs="0"/>
            <xsd:element name="QueueBinding" type="host:QueueDescType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="QueueDescType">
        <xsd:sequence>
            <xsd:element name="QueueDesc" type="xsd:anyType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.5 Host Instance XML Schema

```
<?xml version="1.0"?>
```

```
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host"
xmlns:host="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Host" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
     This schema describes host machine bindings for applications.
      </xsd:documentation>
    </xsd:annotation>
    <!--
  HostDesc is the root element, with any number of applications.
  HostDesc contains a description of a host machine. It contains
  several required fields and can be extended by an arbitrary
  number of parameters.  Optionally, you can provide commands
  for remote copy and execution (rsh, rcp, globusrun).  This must
  be set up externally.

  Hosts also contain one or more bindings for queue execution.  This
  could be a queuing system (PBS, GRD) or it could be
  -->
    <xsd:element name="HostInstance" type="host:HostDescType"/>
    <xsd:complexType name="HostDescType">
        <xsd:sequence>
            <xsd:element name="HostName" type="xsd:string"/>
            <xsd:element name="HostIP" type="xsd:string"/>
            <xsd:element name="WorkDir" type="xsd:string"/>
            <xsd:element name="QueueType" type="xsd:string"/>
            <xsd:element name="QsubPath" type="xsd:string"/>
            <xsd:element name="ExecPath" type="xsd:string"/>
            <xsd:element name="RemoteCopy" type="xsd:string" minOccurs="0"/>
            <xsd:element name="RemoteExec" type="xsd:string" minOccurs="0"/>
            <xsd:element name="QueueBindingInstance" type="host:QueueBindingType"/>
            <xsd:element name="HostParameter" type="host:ParameterType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <!--
  Extend to include host information from QueueInstance.xml
  -->
    <xsd:complexType name="QueueBindingType">
        <xsd:sequence>
            <xsd:element name="QueueDescIns" type="xsd:anyType"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.6 Queue Descriptor XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue"
xmlns:queue="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
     This schema describes queuing systems on a host.
      </xsd:documentation>
    </xsd:annotation>
    <!--
  QueueDesc is the root element and describes a queuing system
  parameters.  An instance of this schema describes a particular
  queuing system (such as PBS).  Queue systems that need
  additional values not explicitly stated in the
  -->
```

```
    <xsd:element name="QueueDesc" type="queue:QueueDescType"/>
    <xsd:complexType name="QueueDescType">
        <xsd:sequence>
            <xsd:element name="MemoryOptions" type="queue:OptionType"/>
            <xsd:element name="JobName" type="queue:OptionType"/>
            <xsd:element name="NumberOfCPUs" type="queue:OptionType"/>
            <xsd:element name="Walltime" type="queue:OptionType"/>
            <xsd:element name="Email" type="queue:EmailOptType"/>
            <xsd:element name="QueueOptions" type="queue:OptionType" minOccurs="0"/>
            <xsd:element name="AccountOptions" type="queue:OptionType" minOccurs="0"/>
            <xsd:element name="QueueParameter" type="queue:ParameterType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--
  This allows us to describe the queue's memory options (ie
  1 GB, 4 GB, etc.)
  -->
    <xsd:complexType name="EmailOptType">
        <xsd:sequence>
            <xsd:element name="JobBeginEmail" type="xsd:string" minOccurs="0"/>
            <xsd:element name="JobEndEmail" type="xsd:string" minOccurs="0"/>
            <xsd:element name="JobAbortEmail" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <!--
  This allows different subqueues to be supported (ie default,
  priority, large, medium, small).
  -->
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="OptionType">
        <xsd:sequence>
            <xsd:element name="DirectiveFlag" type="xsd:string"/>
            <xsd:choice>
                <xsd:element name="UnspecifiedValue" type="xsd:string"/>
                <xsd:element name="PresetsValue" type="xsd:string" maxOccurs="unbounded"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.7 Queue Instance XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:queue="http://grids.ucs.indiana.edu:8005/GCWS/Schema/Queue">
    <xsd:annotation>
        <xsd:documentation>
      This schema describes queuing systems on a host.
       </xsd:documentation>
    </xsd:annotation>
    <!--
  QueueInstance is the root element and describes a queuing system
  parameters.  An instance of this schema describes a particular
  queuing system (such as PBS).  Queue systems that need
  additional values not explicitly stated in the
  -->
    <xsd:element name="QueueInstance" type="queue:QueueInstanceType"/>
    <xsd:complexType name="QueueInstanceType">
        <xsd:sequence>
            <xsd:element name="MemoryOption" type="queue:OptionType"/>
```

```
            <xsd:element name="JobName" type="queue:OptionType"/>
            <xsd:element name="NumberOfCPUs" type="queue:OptionType"/>
            <xsd:element name="Walltime" type="queue:OptionType"/>
            <xsd:element name="Email" type="queue:EmailOptType"/>
            <xsd:element name="QueueOptions" type="queue:OptionType" minOccurs="0"/>
            <xsd:element name="AccountOptions" type="queue:OptionType" minOccurs="0"/>
            <xsd:element name="QueueParameter" type="queue:ParameterType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ParameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="Name" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="OptionType">
        <xsd:sequence>
            <xsd:element name="DirectiveFlag" type="xsd:string"/>
            <xsd:element name="DirectiveValue" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="EmailOptType">
        <xsd:sequence>
            <xsd:element name="JobBeginEmail" type="xsd:string" minOccurs="0"/>
            <xsd:element name="JobEndEmail" type="xsd:string" minOccurs="0"/>
            <xsd:element name="JobAbortEmail" type="xsd:string" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.8 Negotiation XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.gatewayportal.org/wsdl/Negotiate" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:nego="http://www.gatewayportal.org/wsdl/Negotiate" elementFormDefault="unqualified"
attributeFormDefault="unqualified">
    <xsd:complexType name="portType">
        <xsd:sequence>
            <xsd:element name="Operation" type="nego:operationType"/>
            <xsd:element name="ParameterBinding" type="nego:parameterBidingType"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:element name="NegoPortType">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="nego:portType"/>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="parameterBidingType">
        <xsd:sequence>
            <xsd:element name="ParameterDesc" type="xsd:anyType"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:complexType name="operationType">
        <xsd:sequence>
            <xsd:element name="NegoMessageType" type="xsd:anyURI"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:schema>
```

B.9 Negotiation for protocol XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.gatewayportal.org/wsdl/Negotiate/Protocol"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:protocol="http://www.gatewayportal.org/wsdl/Negotiate/Protocol"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xsd:element name="Protocol" type="protocol:protocolType"/>
    <xsd:complexType name="protocolType">
        <xsd:sequence>
            <xsd:element name="NeoParameter" type="protocol:negoparameterType" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="negoparameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:schema>
```

B.10 Negotiation for version XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.gatewayportal.org/wsdl/Negotiate/Version"
xmlns:version="http://www.gatewayportal.org/wsdl/Negotiate/Version" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
    <xsd:complexType name="versionType">
        <xsd:sequence>
            <xsd:element name="NegoParameter" type="version:negoparameterType" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
    </xsd:complexType>
    <xsd:element name="Version">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="version:versionType"/>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="negoparameterType">
        <xsd:sequence>
            <xsd:element name="Value" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:schema>
```

B.11 Fault data XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults"
xmlns:gem="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation>
    Defines a GEM fault.  We organize the fault into two views:
    the "map view", or the projection of the fault on the surface,
    and the "Cartesian view", which describes the fault in in
```

```
    retangular coordiates.
</xsd:documentation>
    </xsd:annotation>
    <xsd:element name="Fault" type="gem:FaultType"/>
    <xsd:complexType name="FaultType">
        <xsd:sequence>
            <xsd:element name="FaultName" type="xsd:string"/>
            <xsd:element name="MapView" type="gem:MapViewType"/>
            <xsd:element name="CartView" type="gem:CartViewType"/>
            <xsd:element name="MaterialProps" type="gem:MaterialPropsType" minOccurs="0"/>
            <xsd:choice>
                <xsd:element name="Slip" type="gem:SlipType"/>
                <xsd:element name="Rate" type="gem:RateType"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="MapViewType">
        <xsd:annotation>
            <xsd:documentation>
    Strike angle is the only interesting parameter for now.  Latitude
    and longitude are fixed to (0,0) but in future should define
    a restricted simple type to allow other values.
</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Latitude" type="xsd:float" fixed="0.0" minOccurs="0"/>
            <xsd:element name="Longitude" type="xsd:float" fixed="0.0" minOccurs="0"/>
            <xsd:element name="StrikeAngle">
                <xsd:simpleType name="Angle">
                    <xsd:restriction base="xsd:float">
                        <xsd:minInclusive value="-360.0"/>
                        <xsd:maxInclusive value="360.0"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CartViewType">
        <xsd:annotation>
            <xsd:documentation>
    Defines the location of the fault in a Cartesian space.
    Arguably could be generated from general coordinate markups.
</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Location" type="gem:LocationType"/>
            <xsd:element name="DipAngle" type="gem:FaultParamType"/>
            <xsd:element name="FaultDimension" type="gem:FaultDimensionType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LocationType">
        <xsd:annotation>
            <xsd:documentation>
    Defines the location of the fault.
</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="XCoordinate" type="gem:FaultParamType"/>
            <xsd:element name="YCoordinate" type="gem:FaultParamType"/>
            <xsd:element name="Depth" type="gem:FaultParamType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FaultDimensionType">
        <xsd:annotation>
            <xsd:documentation/>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Width" type="gem:FaultParamType"/>
            <xsd:element name="Length" type="gem:FaultParamType"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
    <xsd:complexType name="MaterialPropsType">
        <xsd:annotation>
            <xsd:documentation>
    Here just collect the Lame parameters.
  </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="Lame" type="gem:LameType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="LameType">
        <xsd:sequence>
            <xsd:element name="lambda" type="xsd:float"/>
            <xsd:element name="mu" type="xsd:float"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SlipType">
        <xsd:annotation>
            <xsd:documentation>
    These are the components of the displacement (slip) vector
    for the fault.  "Tensile slip" is the displacement of the
    fault perpendicular to its plane, and is usually zero.
  </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="StrikeSlip" type="gem:FaultParamType"/>
            <xsd:element name="DipSlip" type="gem:FaultParamType"/>
            <xsd:element name="TensileSlip" type="gem:FaultParamType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="RateType">
        <xsd:annotation>
            <xsd:documentation>
    These are the components of the displacement (slip) vector
    for the fault.  "Tensile slip" is the displacement of the
    fault perpendicular to its plane, and is usually zero.
  </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="SlipRate" type="gem:FaultParamType"/>
            <xsd:element name="DipRate" type="gem:FaultParamType"/>
            <xsd:element name="TensileRate" type="gem:FaultParamType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="FaultParamType">
        <xsd:annotation>
            <xsd:documentation>
    This type is used to hold both the value and the error.
    Error is optional.  Constrained means the value may
    be described as fixed or allowed to change
    (which is used by simplex).
  </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="value" type="xsd:float"/>
            <xsd:element name="error" type="xsd:float" default="0.0" minOccurs="0"/>
            <xsd:element name="constrained" type="xsd:boolean" default="true" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.12 Displacement data XML Schema

```
<?xml version="1.0"?>
```

```
<xsd:schema targetNamespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:disp="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement">
    <xsd:annotation>
        <xsd:documentation>
    Defines input and output for disloc code.
  </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType name="changeDispType">
        <xsd:sequence>
            <xsd:element name="XCalc" type="xsd:float" minOccurs="0"/>
            <xsd:element name="XResidual" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YCalc" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YResidual" type="xsd:float" minOccurs="0"/>
            <xsd:element name="ZCalc" type="xsd:float" minOccurs="0"/>
            <xsd:element name="ZResidual" type="xsd:float" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="inSARType">
        <xsd:sequence>
            <xsd:element name="Elevation" type="xsd:float"/>
            <xsd:element name="Azimuth" type="xsd:float"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="DisplacementList" type="disp:DisplacementListType"/>
    <xsd:complexType name="DisplacementListType">
        <xsd:sequence>
            <xsd:element name="Displacement" type="disp:DisplacementType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DisplacementType">
        <xsd:sequence>
            <xsd:element name="Displacement3D" type="disp:Vector3DType"/>
            <xsd:element name="Location2D" type="disp:Vector2DType"/>
            <xsd:element name="InitDispParam" type="disp:changeDispType" minOccurs="0"/>
            <xsd:element name="AdjustDispParam" type="disp:changeDispType" minOccurs="0"/>
            <xsd:element name="ObservType" type="xsd:int" minOccurs="0"/>
            <xsd:element name="inSAR" type="disp:inSARType" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Vector3DType">
        <xsd:sequence>
            <xsd:element name="XComp" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YComp" type="xsd:float" minOccurs="0"/>
            <xsd:element name="ZComp" type="xsd:float" minOccurs="0"/>
            <xsd:element name="XError" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YError" type="xsd:float" minOccurs="0"/>
            <xsd:element name="ZError" type="xsd:float" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Vector2DType">
        <xsd:sequence>
            <xsd:element name="XComp" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YComp" type="xsd:float" minOccurs="0"/>
            <xsd:element name="XError" type="xsd:float" minOccurs="0"/>
            <xsd:element name="YError" type="xsd:float" minOccurs="0"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

B.13 Disloc data XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Disloc"
xmlns:disp="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:gem="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults"
xmlns:disloc="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Disloc">
    <xsd:import namespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement"
schemaLocation="Displacement.xsd"/>
    <xsd:import namespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults" schemaLocation="Faults.xsd"/>
    <xsd:annotation>
        <xsd:documentation>
    Defines input and output for disloc code.
  </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="DislocData" type="disloc:DislocDataType"/>
    <xsd:complexType name="DislocDataType">
        <xsd:sequence>
            <xsd:element name="DislocFaultParams" type="disloc:InputType"/>
            <xsd:element name="DislocDisplacement" type="disp:DisplacementType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="InputType">
        <xsd:annotation>
            <xsd:documentation>
    Defines the input for disloc.  Can be translated into
    appropriate format by bridge program.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="DislocFault" type="gem:FaultType" maxOccurs="unbounded"/>
            <xsd:element name="DislocObsv" type="disloc:DislocObsvType" minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="DislocObsvType">
        <xsd:annotation>
            <xsd:documentation>
    Defines the input for disloc.  Can be translated into
    appropriate format by bridge program.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="GridSpec" type="disloc:GridSpecType"/>
                <xsd:element name="PointSpecBag" type="disloc:PointSpecBagType"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="GridSpecType">
        <xsd:annotation>
            <xsd:documentation>
    Defines a regular grid of observation points.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="XSpan" type="disloc:SpanType"/>
            <xsd:element name="YSpan" type="disloc:SpanType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PointSpecBagType">
        <xsd:annotation>
            <xsd:documentation>
    Just a simple holder for pointspecs.  Needed because of
    the choice element in DislocObsvType.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="PointSpec" type="disloc:PointSpecType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="PointSpecType">
        <xsd:annotation>
            <xsd:documentation>
    Defines (x,y) coordinates of observation points.
    </xsd:documentation>
```

```
            </xsd:annotation>
            <xsd:sequence>
                <xsd:element name="xcoord" type="xsd:float"/>
                <xsd:element name="ycoord" type="xsd:float"/>
            </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="SpanType">
            <xsd:sequence>
                <xsd:element name="start" type="xsd:float"/>
                <xsd:element name="stepsize" type="xsd:float"/>
                <xsd:element name="nsteps" type="xsd:int"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

B.14 Simplex data XML Schema

```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Simplex"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:gem="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults"
xmlns:disp="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement"
xmlns:simplex="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Simplex">
    <xsd:import namespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Displacement"
schemaLocation="Displacement.xsd"/>
    <xsd:import namespace="http://commgrids.indiana.edu/GCWS/Schema/GEMCodes/Faults" schemaLocation="Faults.xsd"/>
    <xsd:annotation>
        <xsd:documentation>
    Defines input and output for simplex.
  </xsd:documentation>
    </xsd:annotation>
    <xsd:complexType name="CodeFaultParamType">
        <xsd:sequence>
            <xsd:element name="FaultParamNum" type="xsd:int"/>
            <xsd:element name="ActiveParam" type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:element name="SimplexData" type="simplex:SimplexDataType"/>
    <xsd:complexType name="SimplexDataType">
        <xsd:annotation>
            <xsd:documentation>
    Defines the input for simplex.  Can be translated into
    appropriate format by bridge program.  Input consists
    of input faults, observation points and types, and
    code specific parameters.
    </xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="SimplexFaultParams" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="SimplexFault" type="gem:FaultType"/>
                        <xsd:element name="CodeFaultParam" type="simplex:CodeFaultParamType" minOccurs="0"
maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="SimplexDisplacement" type="disp:DisplacementType" maxOccurs="unbounded"/>
            <xsd:choice>
                <xsd:element name="CodeParams" type="simplex:CodeParamsTypes"/>
                <xsd:element name="BestChi2" type="xsd:double"/>
            </xsd:choice>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CodeParamsTypes">
        <xsd:sequence>
```

```
                <xsd:element name="startTemp" type="xsd:float"/>
                <xsd:element name="max_iter" type="xsd:int"/>
            </xsd:sequence>
        </xsd:complexType>
</xsd:schema>
```

Bibliography

[1] I. Foster, C. Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a New Computing Infrastructure*, Foster I., and Kesselman, C.,eds. Morgan Kauffmann, 1999.

[2] The Globus Toolkit ™ 2.2.2.

See http://www.nsf-middleware.org/NMIR2/components/globus.asp.

[3] Globus: Globus Toolkit 3.0 Fact Sheet.

See http://www.globus.org/toolkit/gt3-factsheet.html.

[4] Legion: A Worldwide Virtual Computer. Andrew Grimshaw.

See http://legion.virginia.edu/.

[5] A. Grimshaw, A. Ferrari, G. Lindahl, K. Holcomb. Metasystems. *Communications of the ACM,* Vol. 41 (11) (1998).

[6] A.S. Grimshaw, A. Natrajan, M.A. Humphrey, M.J. Lewis, A. Nguyen-Tuong, J.F. Karpovich, M.M. Morgan, A.J. Ferrari. From Legion to Avaki: the persistence of vision. In *Grid Computing: Making the Global Infrastructure a Reality*, Berman, F., Fox, G., and Hey, T. Wiley, 2003.

[7] Condor: High Throughput Computing. See http://www.cs.wisc.edu/condor.

[8] D. Thain, T. Tannenbaum, M. Livny. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality*, Berman, F., Fox, G., and Hey, T. Wiley, 2003.

[9] I. Foster, C. Kesselman (Eds.). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.

[10] G. Fox, W. Furmanski. High Performance commodity computing. In *The Grid: Blueprint for a New Computing Infrastructure*. Foster I, Kesselman C(eds.). Morgan Kauffmann, 1999.

[11] Special issue on Grid Computing Environments. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1035-1600 (2002).

[12] W.E. Johnston, D. Gannon, B. Nitzberg. Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid. *Proceedings 8th IEEE International Symposium on High Performance Distributed Computing*, 1999.

[13] M.P. Thomas, J.R. Boisseau. Building Grid computing portals: the NPACI Grid portal toolkit. In *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey. John Wiley & Sons, Chichester, England, March 2003.

[14] K. Schuchardt, B. Didier, G. Black. Ecce--a problem-solving environment's evolution toward Grid services and a Web architecture. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1221-1239 (2002).

[15] D.W. Erwin. UNICORE—a Grid computing environment. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1395-1410 (2002).

[16] Gateway Project. See http://www.gatewayportal.org.

[17] M.E. Pierce, C. Youn, G.C. Fox. The Gateway Computational Web Portal. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1411-1426 (2002).

[18] T. Haupt, E. Akarsu, G. Fox, C. Youn. The Gateway System: Uniform Web Based Access to Remote Resources. *Concurrency: Practice and Experience*, 2000; **12**(8);629-642.

[19] C. Buru, A. Rajasekar, M. Wan. The SDSC Storage Resource Broker. *In Proceedings of CASCON'98*, Toronto, Canada, November 1998.

[20] T. Haupt, P. Bangalore, and G. Henley. Mississippi Computational Web Portal. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1275-1287 (2002).

[21] M. Pierce, C. Youn, O. Balsoy, G. Fox, S. Mock, K. Mueller. Interoperable Web Services for Computational Portals. In *Proceedings of Supercomputing 2002* Baltimore, November 2002.

[22] S. Graham, S. Simeonov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, R. Neyama. Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI. SAMS, Indianapolis, 2002.

[23] Saganich, A., Java and Web Services Primer.
See http://www.onjava.com/pub/a/onjava/2001/08/07/webservices.html.

[24] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Global Grid Forum Draft 2.9 (June 22, 2002).*
See http://www.ggf.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf.

[25] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note 08 May 2000. Available from http://www.w3.org/TR/SOAP/.

[26] K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, S. Pharies. Web Service Inspection Language (WS-Inspection) 1.0. IBM and Microsoft November 2001. Available from http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html.

[27] T. Bellwood, L. Clemont, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, C. von Riegen. UDDI Version 3.0 UDDI Spec Technical Committee Specification. 19 July 2002. Available from http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf.

[28] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana. Web Service Description Language (WSDL) version 1.1. W3C Note 15 March 2001. Available from http://www.w3c.org/TR/wsdl.

[29] CORBA/IIOP Specification.

See http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[30] Distributed Component Object Model (DCOM).

See http://www.microsoft.com/com/tech/DCOM.asp.

[31] Java Remote Method Invocation (RMI). See http://java.sun.com/products/jdk/rmi/.

[32] I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of Supercomputer Applications 2001; 15(3):200-222.

[33] Globus Resource Allocation Manager (GRAM). Documentation available from http://www-unix.globus.org/api/c-globus-2.2/globus_gram_documentation/html/index.html.

[34] Communication in the Globus Toolkit.

See http://www.globus.org/toolkit/communication.html

[35] The Monitoring and Discovery Service (MDS) in the Globus Toolkit.

See http://www.globus.org/mds/mds2/

[36] Grid Security Infrastructure (GSI) in the Globus Toolkit.

See http://www.globus.org/security/

[37] S. Agrawal, J. Dongarra, K. Seymour, S. Vadhiyar. NetSolve: past, present, and future – a look at a Grid enabled server. In *Grid Computing: Making the Global Infrastructure a Reality*, Berman, F., Fox, G., and Hey, T. Wiley, 2003.

[38] H. Nakada, Y. Tanaka, S. Matsuoka, S. Sekiguchi. Ninf-G: a GridRPC system on the Globus toolkit. In *Grid Computing: Making the Global Infrastructure a Reality*, Berman, F., Fox, G., and Hey, T. Wiley, 2003.

[39] Apache Axis. Available from http://xml.apache.org/axis/.

[40] A. Abdelnur, S. Hepper. JSR 168: Portlet Specification. Java Community Process. See http://www.jcp.org/en/jsr/detail?id=168.

[41] Apache Jetspeed Portal. See http://jakarta.apache.org/jetspeed/site/index.html.

[42] A. Kropp, C. Leue, R. Thompson. Web Services for Remote Portlets Specification. Committee Specification 1.0, OASIS 14 July 2003. See http://www.oasis-open.org/committees/download.php/2877/wsrp-specification-1.0-cs-1.0-rev2.pdf.

[43] G. von Laszewski, J. Gawor, P. Lane, N. Rehn, M. Russell. Features of the Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1045-1055 (2002).

[44] J. Novotny. The Grid Portal Development Kit. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1129-1144 (2002).

[45] A. Natrajan, A. Nguyen-Tuong, M.A. Humphrey, M. Herrick, B.P. Clarke, A.S. Grimshaw. The Legion Grid Portal. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1365-1394 (2002).

[46] GridPort Information Repository (GPIR) alpha.
See http://www.tacc.utexas.edu/grid/gpir/

[47] CORBA Security Service.
See http://www.omg.org/technology/documents/formal/security_service.htm.

[48] T. Haupt, E. Akarsu, G. Fox, C. Youn. The Gateway System: Uniform Web Based Access to Remote Resources. *Concurrency: Practice and Experience*, 2000; **12**(8);629-642.

[49] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, G. Premchandran, WebFlow-A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing." *Concurrency: Practice and Experience*, 1997; **9**(6);555-577.

[50] E. Akarsu. *Integrated Three-Tier Architecture for High-Performance Commodity Metacomputing*. Ph. D. Dissertation, Syracuse University, 1999.

[51] J. Wood. *Collaborative Visualization*. PhD Thesis, Leeds University, February 1998.

[52] G. C. Fox, J. Jin. Overview of Collaborative Computing and Some NPAC Experience. ERDC Technical Report May 2000. See http://www.new-npac.org/users/fox/documents/collabcompmay00/collabviz.html.

[53] E. Gamma, R. Helm, R Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley 1995.

[54] Gnuplot. See http://www.ucc.ie/gnuplot/gnuplot.html.

[55] MATLAB. See http://www.mathworks.com/.

[56] R. Clarke. Conventional public key infrastructure: An artefact ill-fitted to the needs of the information society. In *Proceedings of European Conference on Information Systems (ECIS)*, June 2001.

[57] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, *IEEE Communications*, 32(9):33-38. September 1994.

[58] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security, pp. 83-92, 1998.

[59] A. Freier, P. Karlton, P. C. Kocher. The SSL Protocol Version 3.0. Internet Draft(1996). Available from http://wp.netscape.com/eng/ssl3/3-SPEC.HTM.

[60] Adiron: Secure System Design. See http://www.adiron.com.

[61] DeveloperWorks: Web services.
See http://www-106.ibm.com/developerworks/webservices/.

[62] Web Services Developer Center Home.
See http://msdn.microsoft.com/webservices/default.aspx.

[63] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, P. Vanderbilt. Open Grid Services Intrastructure (OGSI) Version 1.0. Draft paper, April 5, 2003. See http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.

[64] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM Software Group, May 2001. See http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

[65] The Castor Project. See http://castor.exolab.org/.

[66] J. J. Barton, S. Thatte, H. F. Nielsen. SOAP Messages with Attachments. W3C Note 11 December 2000. Available from http://www.w3.org/TR/SOAP-attachments.

[67] JavaBeans Activation Framework.
Available from http://java.sun.com/products/javabeans/glasgow/jaf.html.

[68] JXPath. Available from http://jakarta.apache.org/commons/jxpath/.

[69] XML database, Xindice. Available from http://xml.apache.org/xindice/.

[70] S. Mock, K. Mueller, M. Pierce, C.Youn, G. Fox, M. Thomas. A Batch Script Generator Web Service for Computational Portals. *Proceedings of Communications*

in Computation (CIC-02). International Multiconference on Computer Science, June 2002.

[71] Simplified Wrapper and Interface Generator (SWIG). See http://www.swig.org/.

[72] Vaughan-Nichols, and Stephen J. Web Services: Beyond the Hype. *IEEE Computer*. Vol 35, No. 2, p 18-21

[73] G. C. Fox, D. Gannon, M. Thomas. A Summary of Grid Computing Environments. *Concurrency and Computation: Practice and Experience*, Vol. 14, No. 13-15, pp 1035-1044.

[74] G. Fox, H. Bulut, K. Kim, S. Ko, S. Lee, S. Oh, S. Pallickara, X. Qiu, A. Uyar, M. Wang, W. Wu. Collaborative Web Services and Peer-to-Peer Grids. Presented at 2003 Collaborative Technologies Symposium (CTS'03).

[75] S. Carmody. Shibboleth Overview and Requirements. (2001). Available from http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-requirements-01.html.

[76] GSI SOAP. See http://doesciencegrid.org/Grid/projects/soap/.

[77] P. Hallam-Baker and E. Maler. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML). OASIS Standard, 5 November 2002. Available from http://www.oasis-open.org/committees/download.php/1371/oasis-sstc-saml-core-1.0.pdf.

[78] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon. Web Services Security (WS-Security). Version 1.0. 05 April 2002.

See http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.

[79] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, A. Essiari. Certificate-based Access Control for Widely Distributed Resources. *Proceedings of the Eight Usenix Security Symposium*. 1999.

[80] Akenti: Distributed Access Control. Accessed from http://www-itg.lbl.gov/Akenti/.

[81] J. Linn. Generic Security Services Application Program Interface, Version 2. OpenVision Technologies, January 1997.
See http://www.rfc-editor.org/rfc/rfc2078.txt.

[82] G. Fox, S. Pallickara. The Narada Event Brokering System: Overview and Extensions. Proceedings of the *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 2002. pp 353-359.

[83] Java Secure Socket Extension (JSSE) Reference Guide. See http://www.ssw.uni-linz.ac.at/Services/Docs/JDK/guide/security/jsse/JSSERefGuide.html.

[84] M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg. SIP: Session Initiation Protocol. IETF Request for Comments 2543 (1999). Available from http://www.ietf.org/rfc/rfc2543.txt

[85] S. Parameswar, B. Stucker. The SIP Negotiate Method. Internet Draft, Internet Engineering Task Force, June, 2002 Work in progress. Available from http://www.softarmor.com/sipwg/drafts/draft-spbs-sip-negotiate-01.txt

[86] T. Berners-Lee, J. Hendler, O. Lassila. The Semantic Web. May 17, 2001.
See http://www.scientificamerican.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[87] O. Lassila, R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C, February 22 1999. See http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[88] J.Rosenberg, H.Schulzrinne. An Offer/Answer Model with SDP. Internet Draft, Internet Engineering Task Force, February 21, 2002 Work in progress. See http://rfc3264.x42.com/

[89] OpenSAML—An Open Source Security Assertion Markup Language Implementation. See http://www.opensaml.org/.

[90] QuakeSim System Documentation.
See http://www-aig.jpl.nasa.gov/public/dus/quakesim/documentation.html.

[91] A. Donnellan, et al. Final Report on the GESS Study—Inversion of Earthquake Fault Parameters Using Multiple Look Angles. NASA JPL internal report (2002).

[92] A. Donnellan, et al. Elastic Dislocation Fault Parameters.
See http://www.servogrid.org/GCWS/Schema/GEMCodes/disloc.ps.

[93] G. Lyzenga. Simplex Version 4.0 Input File Format.
See http://www.servogrid.org/GCWS/Schema/GEMCodes/simplex4.ps.

[94] J. Clarke, R. R. Namburu. A Distributed Computing Environment for Interdisciplinary Applications. *Currency and Computation: Practice and Experience*, Vol. 14, No. 13-15, p. 1161-1174 (2002).

[95] NCSA HDF Home Page. Available from http://hdf.ncsa.uiuc.edu/.

[96] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, A. Slominski. Grid Web Services and Application Factories. Draft paper. See http://www.extreme.indiana.edu/xgws/afw/appFactory.pdf.

[97] O. Balsoy, Y. Jin, M. Pierce, G. Fox. Automating Metadata Web Service Deployment for Problem Solving Environments. *The 3nd International Conference on Computational Science, ICCS '03,* Melbourne, Australia. June 2-4, 2003.

VITA

NAME OF AUTHOR: Choonhan Youn

PLACE OF BIRTH: KyungJu, Korea

DATE OF BIRTH: October 17, 1968

DEGREES AWARDED:

B.S. in Computer Enginnering, February 1991, University of Ulsan, Ulsan, Korea

M.S. in Computer Engineering, February 1993, University of Ulsan, Ulsan, Korea

M.S. in Computer Engineering, May 1998, Syracuse University, New York, USA