The Ninf Portal
An Automatic Generation Tool for Grid Portals

Toyotaro Suzumura
Tokyo Institute of Technology and Japan Society for the Promotion of Science

Hidemoto Nakada
National Institute of Advanced Industrial Science and Tokyo Institute of Technology

Masayuki Saito
Tokyo Institute of Technology

Satoshi Matsuoka
Tokyo Institute of Technology and National Institute of Informatics

Yoshio Tanaka
National Institute of Advanced Industrial Science

Satoshi Sekiguchi
National Institute of Advanced Industrial Science

ABSTRACT

As the Grid proliferates as the next-generation computing infrastructure, a user interface in the form of "Grid Portals" is becoming increasingly important, especially for computational scientists and engineers. Although several Grid Portal toolkits have been proposed, portal developers still must build and deploy both the user interface and the application, which results in considerable programming efforts. We aim to ease this burden by generating a portal frontend (that constitutes of JSP and Java Servlets) from an XML document for the former, and a GridRPC system, Ninf-G for easily "gridifying" existing applications for the latter, and realizing their seamless integration. The resulting system, which we call the Ninf Portal, allowed concise description and easy deployment of a real Grid application with greatly small programming efforts.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Programming Environments—*Interactive environments*

General Terms

Design, Experimentation

Keywords

Grid Portals, Ninf-G, GridRPC, Portal, JSP, Servlet

1. INTRODUCTION

Computational Grids have emerged as a distributed computing infrastructure for providing pervasive, ubiquitous access to a diverse set of resources ranging from high-performance computers (HPC) to tertiary storage systems to large-scale visualization systems to expensive and unique instruments including telescopes and accelerators. However, Grid infrastructure only provides a common set of services and capabilities that are deployed across resources, and it is the responsibility of the application scientist to devise methods and approaches for accessing Grid services. For this reason, higher-level tools in the form of "Grid Portal" are becoming increasingly important to more effectively take advantage of Grid infrastructure, and in fact HotPage[1] has already been widely used by NPACI users.

A Grid portal is defined as a web based application server enhanced with necessary software to communicate with Grid services and resources, so that users could access to Grid resources from a web browser in a uniform way without the need to install any software on their machine. In general, a Grid portal provides application scientists with a customized view of software and hardware resources specific to their particular problem domain and provides a single point of access to Grid resources they already have been authorized to use.

A Grid portal entails a three tier architecture adopted by general Web applications, consisting of (1) a first tier of clients, (2) a middle tier of brokers or servers, and (3) a third tier of object repositories, compute servers, databases, or any other resource or service needed by the portal. The portal should provide a set of mechanisms to allow users to login to the Grid with single sign-on, invoke Grid applications on remote resources along with a set of data, monitor/control the applications, upload/download the required data. As such, developing a Grid portal needs some extra tasks in addition to implementing basic features in general Web applications.

Some research projects[2, 3] have built a helper tool to build a Grid portal, providing a basic set of components to be implemented in the portal. Although it might be comparatively more useful than building them from the scratch, such toolkits would not be sufficient in that they do not provide mechanisms to help create a frontend user interface for inputting data to be passed to a Grid application, and build the application as backend.

JGI'02, November 3–5, 2002, Seattle, Washington, USA.

In this paper, we propose a Grid portal construction toolkit, what we call Ninf Portal, designed to facilitate the development of Grid portals by automatically generating the portal frontend that constitutes of JSP(JavaServer Pages) and Java Servlets from an XML document and then providing Ninf-G to build Grid applications as the backend, and evaluate the validity by building a Grid portal for the existing Grid application.

The rest of this paper is organized as follows. In Section 2, we review the requirements and general architecture of a Grid portal. In Section 3, we provide an overview and design of the Ninf Portal. In Section 4, we give an overview of the Ninf-G system that is an essential component constituting the Ninf Portal. In Section 5, we give a detailed explanation on the Ninf-Portal front-end. In Section 6, we evaluate the system building a Grid portal for the real Grid application via the use of the Ninf Portal. Section 7 describes the related work. Section 8 gives conclusions and future work.

2. GRID PORTAL REQUIREMENTS

In this section we review the requirements and general architecture of a Grid portal.

2.1 Supported Services

A Grid Portal should provide the following services:

Authentication and Authorization Authentication identifies who is connecting to the server; Authorization is what resources to permit to the user that has been identified. A Grid portal should also provides single sign-on mechanism that allows users to multiple remote resources on the Grid after they once authenticate to the portal, and should assume that users would like to access the portal at a location where their Grid credentials would not normally be available to them.

Uploading and Downloading A Grid portal should provide a mechanism that enables users to upload their requisite input data to remote resources for computation, as well as enables them to download the result data via their web browser in an intuitive way.

Job Management A Grid portal should provide a mechanism which allows users to manage their jobs, i.e., executing their Grid application programs via the web interface in a reliable and secure way, monitoring the status of running jobs and stopping/canceling them if necessary.

Information Service A Grid portal should provide an interface to show information on the Grid, so that users could know what resources are available on the Grid and which resources are matched for their application needs.

2.2 Architecture

In general, a Grid portal is largely divided into the frontend and the backend. The frontend is a web application that shows a user interface for searching suitable computing resources and inputting necessary data for the computation, and handles authentication. Existing frameworks are used as building blocks for the frontend to generate dynamic contents interacting with Web servers, e.g., HTML, CGI, and Java Servlet, etc. Meanwhile, the backend is a Grid application responsible for receiving data from the frontend and performing computations with them on the Grid, and is usually built on top of the Globus Toolkit. Some research projects such as GPDK[3] and HotPage[1], have emerged as a tool to help build the portal frontend, but no systems were designed to facilitate the development of both the frontend and backend.

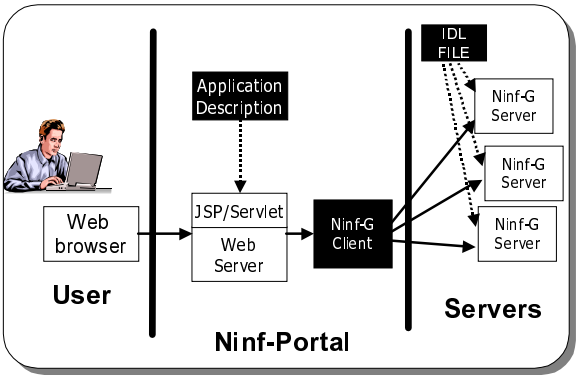

Figure 1: Ninf Portal Architecture

3. THE NINF PORTAL

In this section we first describe an overview and overall architecture of the Ninf Portal. In the subsequent subsections, we illustrate building blocks with which we have implemented our system and then explain how our system enables the automatic user interface generation and single sign-on.

3.1 Overview and Architecture

To address the issues mentioned in the previous section, we propose the Ninf Portal to facilitate the development of a Grid portal, providing a tool to help users to build both the frontend and the backend in a straightforward way. With the Ninf Portal, the frontend consisting of a set of web pages, would be automatically generated from the interface information described in XML; the backend, a Grid application, would be easily developed using Ninf-G to allow users to write the Grid application if they make a local function call. Detailed information on Ninf-G is described in Section 4.

The architecture of the Ninf Portal is shown in Figure 1. The left boxes in the figure is a frontend server that processes most requests by users, such as showing the entry page for the portal, subsequent pages generated by the portal and the user, who initiates all actions in the portal. The frontend invokes the backend which is the right shaded box named `Ninf-G Client` in the figure, along with input data specified in the frontend.

The backend is a Grid application accessing an underlying Grid infrastructure, Ninf-G. The application can be built as a program executed from a command line with input/output data specified as the arguments. Input data are specified either by a string or a filename, and output data are specified either by the standard output or a filename.

3.2 Technologies Employed

Since a portal frontend has the same architecture as general web applications, we may take advantage of various

frameworks widely used for those applications such as CGI (Common Gateway Interface), Java Servlets, JSP, etc. CGI defines the communication semantics between Web servers and application programs, and can be built via the use of Perl. While in practice CGI is available at most of servers, it has some drawbacks in terms of the process execution and session management. The main drawback of CGI is overall inefficiency in handling concurrent client requests. This performance problem comes from extra time taken when CGI needs to create a separate process for each user request, and this process will be terminated as soon as the data transfer is completed.

Meanwhile, Java Servlets is becoming an increasingly popular alternative to traditional CGI. Servlets is a server side Java code that runs in a server application to answer client requests. Servlets is more efficient in terms of performance since it is loaded into the memory when called the very first time, and the servlet remains in memory after the request is processed, and will not be unloaded from memory until the web server is shut down. Furthermore, the other key technology is JavaServer Pages (JSP) that allows you to write Java codes directly in Web pages, resulting in the ability to generate highly dynamic web applications on the fly.

The Ninf Portal employs Java Servlets and JSP as building blocks for the following reason: 1) the session management is simple, 2) Java interface for a set of Globus services, i.e., Java CoG Kit[4], facilitates the access for information services on the Grid. 3) A large variety of Java API enables the integration with other components such as Grid information services.

3.3 Automatic User Interface Generation

In general, a user interface for Web applications is comprised of a set of HTML pages and a servlet capable of handling data. In the Ninf Portal, the only task that portal developers should perform for obtaining the user interface is just describing an XML document, with which the Ninf Portal automatically generates both a JSP file that creates HTML pages on the fly and a general-purpose Java servlet that is capable of handling data.

3.4 Single Sign-On

The authentication in the backend of the Ninf Portal, i.e. Ninf-G, is handled via the security mechanism of the Globus Toolkit, i.e., the GSI protocol. GSI identifies the user's identity by credential signed by user's certificate. The delegation mechanism realizes single sign-on to allow users to access to multiple resources on the Grid, after they once authenticate to the Grid. Consequently, if a Grid portal is capable of retrieving a user's credential in a secure way, single sign-on from the portal would be realized.

The straightforward way of giving a user's credential to a Grid portal is uploading it via a secure HTTP protocol, HTTPS. However, if the private keys are placed on the web server, the entire site security would be greatly threatened by the potential vulnerability of the web server being hacked into and the private keys compromised.

To address these issues, we have introduced an online credentials repository called MyProxy[5], which manages a set of credentials and allow other hosts to retrieve them using username and passphrase. MyProxy has been designed with a great consideration of security such as communication protected by the use of GSI as well as not exposing a private key. The Java Cog Kit provides a client interface to MyProxy and could be easily incorporated into JSP and Servlets with which we have implemented the Ninf Portal.

4. NINF PORTAL BACKEND: NINF-G

Ninf-G is a GridRPC system which we have reimplemented the Ninf system[6] on top of the Globus Toolkit, offering network-based numerical library functionality via the use of RPC technology. Parts of applications making procedure calls can be replaced with high-performance, remote equivalents in a substantially transparent manner, usually only a small modification to the call itself, without any RPC data types, prologue/epilogues, IDL management, etc. Figure 2 shows a snippet of a Ninf-G client program that makes a remote call of matrix multiply on a remote server. Additionally, Ninf-G provides an asynchronous invocation method to exploit network-wide parallelism. For instance, it would be possible to issue a request to Ninf-G, continue with the other computation, and poll for the request later.

A library provider, who provides the numerical library and computational resource to the network at large, would describe the interface of the library function in Ninf IDL. For instance, an interface description for matrix multiplication is shown in Figure 3.

Ninf IDL is an interface description language designated to numerical applications, the supported data type in Ninf-G is tailored for such a purpose; for example, the data types are limited to scalars and their multi-dimensional arrays. On the other hand, there are special provisions in the IDL for numerical applications, such as support for expressions involving input arguments to compute array size, designation of temporary array arguments that need to be allocated on the server side but not transfered, etc.

An interface description is compiled by the Ninf-G interface generator to generate a stub program for each library function described in its interface information. The interface generator also automatically outputs a makefile with which the Ninf-G remote library can be created by linking the stub programs and library functions.

Ninf-G employs a set of Globus services: MDS, GRAM, Globus I/O in order to invoke remote libraries as depicted in Figure 4. The process is as follows. For those of you who are not familiar with the Globus services, we would like you to refer Appendix A.

Querying the argument and executable path to MDS The Ninf-G system employs the MDS server to publish the executable path and argument information of available remote numerical libraries. A client extracts the information from MDS using the library name. The result from MDS could be cached on the client side and reused until it is expired, which reduces extra cost for querying to MDS.

Launching the remote library via GRAM The client library then interprets and marshals the arguments on the stack according to the information supplied in the previous step, and then requests the execution of the remote library to GRAM using the obtained executable path.

Callback from the remote library The remote library, after executed successfully, retrieves the client information including his IP address and port number from

```
double A[N*N], B[N*N], C[N*N];
....
grpc_call("sample/mmul", N, A, B, C);
....
```

Figure 2: Ninf-G Client Program Example

```
Module sample;
Define mmul(IN  int    N,
    IN  double A[N*N],
            IN  double B[N*N],
            OUT double C[N*N])
Required "mmul_lib.o"
Calls "C" mmul(N, A, B, C);
```

Figure 3: Ninf IDL Example

the arguments, and then connects to the client via Globus I/O. The communication between the client and the remote library is continuingly done through the port number. The port number at which the client listens to, performs authentication and authorization so that any programs could not connect to it without having the appropriate certificate. As such, this mechanism reduces security risks including the fact that any third parties could make a connection pretending to be the callback.

5. NINF PORTAL FRONTEND

In section 3, we explained an overview of the overall Ninf Portal. This section focuses on the detailed part of the Ninf Portal frontend.

5.1 Overview

The Ninf Portal frontend handles input from users and displays results from the backend application on the Grid. A user can send data by (1) directly giving a string or (2) uploading a file stored on a user's desktop or (3) directly writing data in the form. The portal then gives him the result by either displaying it on the browser or specifying the URL where it is stored. The Ninf Portal allows users to specify these methods in Grid Application IDL described in the next section.

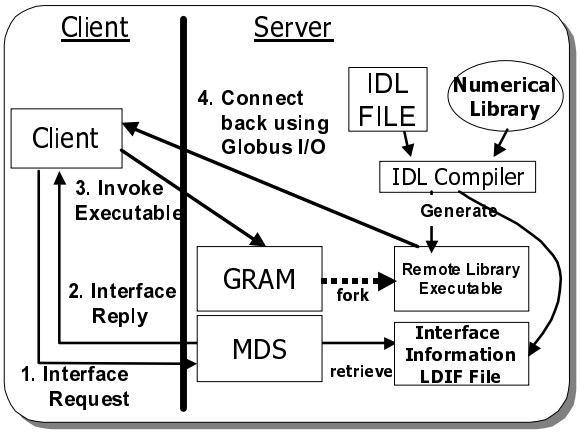

Figure 4: Ninf-G Architecture

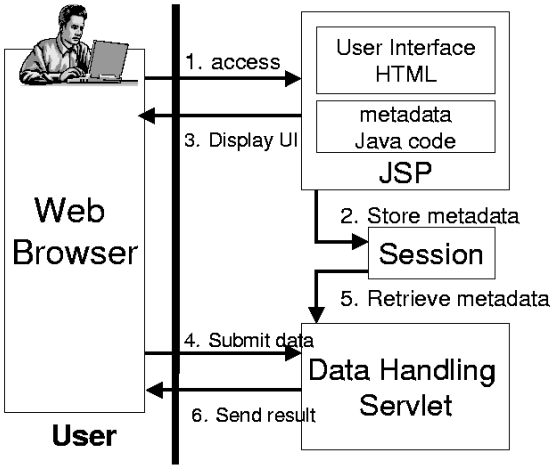

Figure 5: Frontend Architecture

5.2 Architecture

The frontend architecture is depicted in Figure 5, constituting of JSP that displays a user interface and receives input data, and a Java servlet that invokes a backend Grid application along with the data.

JSP and Session A metadata representing a set of input data is required in order for a data handling servlet (described in the next section) to deal with submitted data. A metadata includes a variety of information such as a name of a data filed, a information on which the data is a value or an uploaded data, etc., and will be stored within Java codes of a JSP page.

A JSP page stores a metadata in the session, i.e., cookies for the later usage. After users input necessary data in the field of the HTML page generated by JSP, the request would be delegated to the servlet, which later extracts the metadata from the session, with which the submitted data would be parsed and the backend application, built on Ninf-G, is launched as illustrated in Figure 5.

A session is used to share information for one user across multiple pages while visiting a web site. In other words, a session object is a way of retaining state for a normally stateless HTTP web site. By default, all JSP pages have access to the implicit session object.

Data Handling Servlet A data handling servlet is a general-purpose servlet which handles the following service:

- reading uploaded data
- executing a Grid application
- generating a result page

When executing a Grid application, the user's certificate, retrieved from a MyProxy server at login, must be used to guarantee the security . The servlet would write out the certificate to a temporary directory and set the appropriate permission to avoid eavesdropping. Next, the servlet executes a Grid application after setting an environmental variable to the certification file.

```
<!ELEMENT Application (Information*,
          ArgumentFormat, Argument*)>
<!ELEMENT Information (name?,location?,
          manufacturer?,description?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT location (#PCDATA)>
<!ELEMENT manufacturer (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT ArgumentFormat (#PCDATA)>
<!ELEMENT Argument (type,info,
                    method?,comment?)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT method (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

Figure 6: Grid Application IDL Schema

5.3 Grid Application IDL

Grid Application IDL is used to describe a user interface information of a Grid application in the form of XML and the schema is depicted in Figure 6. The benefit of using XML is that we could use the existing XML parsers as the compiler. We have used a Java-based XML parser that supports the DOM (Document Object Model). In Figure 6, the `location` element defines an executable path of a Grid application; the `ArgumentFormat` element defines a set of arguments to invoke the application; the `Argument` element defines the name and type of each field. Multiple arguments can be described in an arbitrary order.

A JSP page, generated from Grid application IDL, can be largely divided into two parts. The former is HTML codes responsible for handling input from users, and the latter is Java codes responsible for storing metadata that specifies the way of handling submitted data into the session. Furthermore, type checking in the input form is handled by JavaScript. For instance, if you input a floating point data into the field declared as integer, that would give a caution and enable an error detection at an earlier stage.

6. EVALUATION

In this section, we evaluate our system through the experiment of building the Grid portal for the real application, BMI (Bilinear Matrix Inequality) Eigenvalue Problem [8] that is one of optimization problems minimizing the greatest eigenvalue of a bi-linear matrix function, and that is a greatly compute-intensive application as well as could be easily parallelized. Building a Grid portal by the use of the Ninf Portal falls into two steps: the first step is to "gridify" the application using Ninf-G, and the next step is to generate a web interface from a Grid Application IDL. In the preceding section, we give you more concrete description as to what users should do at each step.

6.1 Building Grid Application

Firstly, users should build a Grid application capable of making use of remote computational resources on the Grid, via the use of Ninf-G that would facilitate the development of the Grid application. Three programs are involved to build the Grid application via Ninf-G; a client program, a remote library, and a IDL for a remote numerical library. Therefore, when making an application available on the Grid, users should first divide it into them.

Meanwhile, a web interface communicates with a Grid application by passing a set of information as the arguments. The BMI application takes as input a string representing a type of computation and a data file for computation, and finally prints out the result after the computation.

A client program could be parallelly executed on multiple servers by invoking remote libraries with an asynchronous method called `grpc_call_async`. Although it takes a deal of efforts to control multiple communication channels required for parallel execution, Ninf-G encapsulates its burden so that programmers could build a distributed and parallel application on the Grid in a straightforward way.

6.2 Writing Grid Application IDL

As mentioned in the previous section, the BMI application needs two elements as input: a string representing a type of problem and a file containing a configuration information, and the standard output as output. Therefore, since standard output is set as default for showing the result, users should describe two input information in the Grid application IDL. Figure 7 illustrates the Grid application IDL for BMI.

After the compiler processes the Grid Application IDL file, a JSP file for receiving input from users would be generated. The user interface for the Grid Portal would then be generated by deploying the JSP file at an appropriate directory on the web server. Figure 8 shows the JSP file automatically generated by the compiler. The enclosed part between `<%` and `%>` is Java codes, and the first line loads necessary Java packages related to the application. The following lines below the fifth line, instantiate an object of `PortalApplicationDescription` class representing the metadata of the Grid application, and set it to the attribute of the session object. Using the metadata, the data handling servlet would invoke the application and show the result. The part enclosed between the `form` tags shows the actual input interface. Note that the information enclosed by the `Information` tag in the IDL file is used for the attributes of the `form` tag. As shown in the JSP file, a variety of elements should be carefully described so that the JSP page could communicate with the data handling servlet. This burden for portal developers would be alleviated by creating JSP from Grid Application IDL via the use of the Ninf Portal.

6.3 Execution Example

Next, we illustrate the screenshot of the actual Grid portal in Figure 9 that is generated in the previous sections.

The left window handles the login to authenticate users and the center window handles a set of input data from users. This image would be drawn by JSP generated by Grid Application IDL.

Since `upload` is specified in the `Argument` tag in the Grid Application IDL, the generated JSP has a field for uploading a file. The right window in Figure 9 shows the outputted result after the computation is finished. This sample shows the result directly on the screen, but it would be possible to specify the way to download the result with a file.

7. RELATED WORK

To date, a variety of systems have been proposed as portal construction toolkits, such as GridPort from NPACI[2], Grid Portal Development Kit[3] from NLANR, XCAT Science Portal [9] from Indiana University, etc.

GridPort is the most well-known toolkit widely used for

```
<?xml version="1.0" encoding="shift_jis"?>
<!DOCTYPE application SYSTEM
                "JSPGenerator.dtd">
<Application>
  <!-- Application Information -->
  <Information>
    <name> BMI </name>
    <location>
   /home/saito/work/BMIClientC/BMISolver
    </location>
    <manufacturer>
      Kento Aida
    </manufacturer>
    <appdescription>
      BMI Application Portal
    </appdescription>
  </Information>

  <ArgumentFormat>
    -t $type $uploadfile
  </ArgumentFormat>
  <!-- Arguments Information ------ -->
  <!-- ----- first Argument ------- -->
  <Argument>
    <argname> uploadfile </argname>
    <type>inputfile</type>
    <info> InputFile </info>
    <method> upload </method>
    <description>
      inputfile of data
    </description>
  </Argument>
  <!-- ---- second Argument ------- -->
  <Argument>
    <argname> type </argname>
    <type> string </type>
    <info> type </info>
    <description>
      type of program
    </description>
  </Argument>
</Application>
```

Figure 7: Grid Application IDL Example

building Grid portals. The representative example is HotPage, which provides users with a view of distributed computing resources and allows individual machines to be examined as to status (up or down), load, etc. Besides examining machines, users can access files and perform routine computational tasks. Most of the CGI programs are built in Perl, and users should use Perl when registering a new Grid application. Backend application programs must be built in Globus API, which takes a great deal of efforts for the user.

Grid Development Portal Toolkit (GPDK) is a Java-based toolkit which provides several key reusable components for accessing various Grid services, and provides a customizable interface allowing scientists to perform a variety of Grid operations including remote program submission, file stating, and querying of information services from a single, secure gateway. GPDK give users direct access to Grid services in the form of Java Beans from JSP. However, it is not clear that JSP would be a suitable technology to describe the access to the Grid environment (that is often estimated to take a relatively longer time) because the technology was designed for web applications available for a short period time.

Another very important project is the Indiana/NCSA Science Portal. In this effort, portals are designed using a notebook of typical web pages, input forms, and execution scripts. Notebooks have an interactive script/forms editor based on JPython that allows access to other tool kits

```
<%@ page import="ninfPortal.*" %>
<html>
<head> <title>BMI portal</title></head>
<body>
<%
  int    argnumber = 2;
  String argformat = "-t $type $uploadfile";
  String executablepath =
     "/home/saito/work/BMIClientC/BMISolver";
  String args[]       = {"inputfile","string"};
  String namerow[]    = {"uploadfile","type"};
  String filemethod[] = {"upload","null"};
  PortalApplicationDescription obj =
    new PortalApplicationDescription(
          argnumber, argformat, executablepath,
          args, namerow, filemethod);
  session.setAttribute("inputs",obj);
%>
<br> Welcome to BMI Application Portal! <br>
<form action="/bmi/servlet/Portal"
      name="MyForm" method=post
      ENCTYPE="multipart/form-data"
      onSubmit="return checkData(this)">
 <table border = 3 align = center> <tr>
     <td>InputFile</td>
     <td><input type = file name = arg0></td>
 </tr> <tr>
     <td>type</td>
     <td><input type = text name = arg1></td>
 </tr> </table>
 <center>
   <input type  = submit align = center
          value = "submit">
 </center>
 </form>
</body>
</html>
```

Figure 8: Generated JSP Example

such as CoG Kit and the Common Component Architecture Toolkit (CCAT). The drawback is its high installation cost, with the need to install web server and application logic and access to the Grid with multiple protocols on the client side.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we have described the Ninf Portal system that helps portal developers to build their portal in a intuitive and straightforward way. The system alleviates the user's burden by automatically generating JSP pages as portal frontend from an XML document and utilizing a GridRPC system, Ninf-G to build Grid applications as portal backend. Moreover, we confirmed the high validity of the Ninf Portal system by practically building the Grid portal for the real Grid application via the use of the system. Future work on the system includes the following tasks:

Integrating Information Service: We do not provide an information service which is an essential part of Grid portals since we focus on the simplicity of program execution from the portal. Such a service is useful for obtaining both static and dynamic information on software and hardware resources. We must incorporate the service into the Ninf Portal, as well as provide a mechanism that allows application users to view their runs as they progress.

Building Java APIs for Ninf-G: The current implementation of Ninf-G has supported only C client interface. The support for Java client interface is inevitable since users could integrate Grid applications built by the

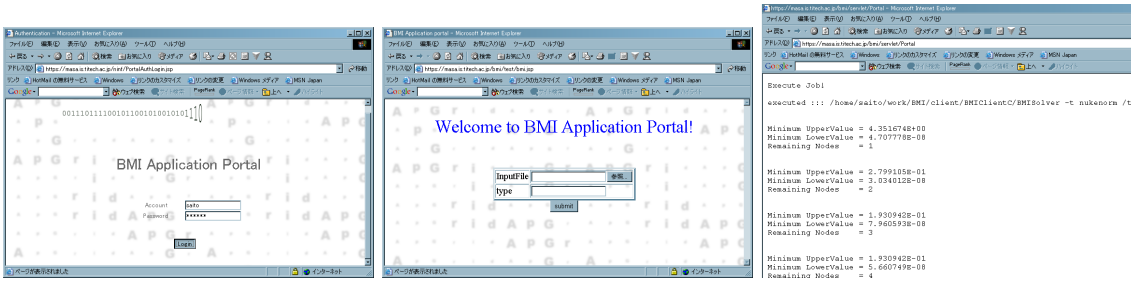

Figure 9: Execution Example

Java interface, with Java Servlets on the server side. This would allow users to install the portal in more straightforward way and would avoid the entire site security hole greatly threatened by the vulnerability with the private keys stored out of the web server.

Support for Scripting Interface: Currently, the Ninf Portal only allows users to use Grid applications offered by portal developers, but the ability to manipulate a Grid application in the context of a portal interface and architecture would be extremely important. One solution would be the mechanism adopted by Indiana XCAT Portal, that incorporates a scripting interface into the portal so that users could build a Grid application from the web browser. In the case of the Ninf Portal, this could be done by the scripting interface wrapped up with the Ninf-G Java API, and we need to consider possible security issues since this mechanism would lead to the security vulnerability, with any malicious codes runnable on the portal.

APPENDIX

A. GLOBUS

In this appendix, we briefly review a set of Globus services that we exploit in the Ninf-G system, i.e. GRAM, MDS, Globus I/O, for those who are not familiar with these technologies.

GRAM The Globus Resource Allocation Manager(GRAM) supports remote submission of a computational request to a remote computational resource, and subsequent monitoring and control of the resulting computation. GSI security mechanisms are used in all operations to authenticate the requestor and for authorization. Authentication is performed using the supplied proxy credential, hence providing for single sign-on. Authorization implements local policy and may involve mapping the user's "Grid id" into a local subject name; however, this mapping is transparent to the user. Ninf-G employs GRAM to execute a numerical library in a secure manner.

MDS The Globus Toolkit's MDS provides basic mechanisms for discovering and disseminating information about the structure and state of Grid resources. The Monitoring and Discovery Service(MDS) is the information services component of the Globus Toolkit. MDS uses an extensible framework for managing static and dynamic information about the status of a computational grid and all its components: networks, compute nodes, storage systems, and instruments. The MDS uses the Lightweight Directory Access Protocol (LDAP) as a uniform interface to such information. Ninf-G publishes a variety of function interfaces and allows clients to retrieve them.

Globus I/O Globus I/O is a secure communication module which provides a uniform I/O interface to stream and datagram style communications. The module provides a service to support nonblocking I/O and handle asynchronous file and network events. Ninf-G employs the module to enable the communication between the client and the executable at the server side.

B. ADDITIONAL AUTHORS

C. REFERENCES

[1] NPACI HOTPAGE, https://hotpage.npaci.edu/.

[2] Thomas, M., Mock, S. and Boisseau, J.: Development of Web Toolkits for Computational Science Portals: The NPACI HotPage, *Proceedings of HPDC 9*, pp. 308–309 (2000).

[3] The Grid Portal Development kit, http://dast.nlanr.net/Projects/GridPortal/.

[4] von Laszewski, G., Foster, I. and Gawor, J.: CoG Kits: A Bridge Between Commodity Distributed Computing and High-Performance Grids,A Java Commodity Grid Kit, *ACM 2000 Java Grande Conference* (2000).

[5] Novotny, J., Tuecke, S. and Welch, V.: Initial Experiences with an Online Certificate Repository for the Grid: MyProxy, *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press (2001).

[6] Nakada, H., Sato, M. and Sekiguchi, S.: Design and Implementations of Ninf: towards a Global Computing Infrastructure, *Future Generation Computing Systems, Metacomputing Issue*, Vol. 15, No. 5-6, pp. 649–658 (1999).

[7] Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications* (1997).

[8] Kento Aida, Yoshiaki Futakata, Shinji Hara: High-performance Parallel and Distributed Computing

for the BMI Eigenvalue Problem, *Proc. 16th IEEE International Parallel and Distributed Processing Symposium, 2002* .

[9] Krishnan, S., Bramley, R., Gannon, D., Govindaraju, M., Indurkar, R., Slominski, A. and Temko, B.: The XCAT Science Portal, *Supercomputing 2001* (2001).