# A *Jini-based* Computing Portal System

Toyotaro Suzumura
Tokyo Institute of Technology
2-12-1 Ookayama Meguro
Tokyo, Japan, 152-8552
suzumura@is.titech.ac.jp

Satoshi Matsuoka
Tokyo Institute of Technology/JST
2-12-1 Ookayama Meguro
Tokyo, Japan, 152-8550
matsu@is.titech.ac.jp

Hidemoto Nakada
AIST/Tokyo Institute of Technology
1-1-1 Chu-o Daiichi Higashi
Tsukuba-shi
Ibaraki, Japan, 305-8561
nakada@aist.go.jp

## ABSTRACT

JiPANG(A Jini-based Portal Augmenting Grids) is a portal system and a toolkit which provides uniform access interface layer to a variety of Grid systems, and is built on top of Jini distributed object technology. JiPANG performs uniform higher-level management of the computing services and resources being managed by individual Grid systems such as Ninf, NetSolve, Globus, etc. In order to give the user a uniform interface to the Grids JiPANG provides a set of simple Java APIs called the JiPANG Toolkits, and furthermore, allows the user to interact with Grid systems, again in a uniform way, using the JiPANG Browser application. With JiPANG, users need not install any client packages beforehand to interact with Grid systems, nor be concerned about updating to the latest version. Such uniform, transparent services available in a ubiquitous manner we believe is essential for the success of Grid as a viable computing platform for the next generation.

## 1. INTRODUCTION

Web portals such as Yahoo now provide transparent and ease-of-use interface to a complex set of web pages on the Internet, and have led to the explosive growth of the web. Recent developments of a variety of Grid systems such as Globus[14, 13, 22], NetSolve[10], and Ninf[20, 21], have opened up opportunities for sharing computing resources including remote scientific instruments and storage devices and high-performance computing services, as well as sharing computed and instrumented information as the web has achieved. However, in order for scientists to use the Grid effectively as a problem solving infrastructure, transparent and easy-of-use interfaces to comparatively more complex set of Grid services are necessary, just as web portals have achieved for the web. Such portal-like systems for the Grid are now referred to as "computing portals", and there are a variety of on-going projects such as CoG[15], GridPort[18], and WebFlow[12], and in fact presently there is a dedicated GCE working group within the Global Grid Forum[2]. However, existing efforts only allow users to access specific Grid systems or resources, and rely on the portal system to be continuously updated along with the upgrade of the Grid system. Due to the continuous and increased development of the Grid infrastructure, such updates will likely to be frequent, and could serve a negative effect towards widespread adaption of the Grid.

The web portal could also encounter such problems; the reason for their success is that software infrastructure to support their evolution has embodied flexible and dynamic charactersitics to support the underlying change. On the client side it is the use of HTML/XML, JavaScrtipt, Java, etc., and on the server side, the combined use of CGI with scripting languages, Java Servlets, and component technologies. What software technologies would a computing portal then require in order to achieve similar characteristics, i.e., seamless, uniform, and intuitive interface to multiple software infrastructure on the Grid, and support for their rapid evolution ?

We are currently building a computing portal system called JiPANG whose aim is to satisfy such goals. JiPANG is built on top of Java Jini technology, and copes with dynamicity of the Grid infrastructure in a portable manner. JiPANG augments Jini so that it satisfies properties important for the Grid such as scalability, as well as offering a software framework in the form of toolkits and various tools that makes it easier for Grid builders to integrate existing Grid services, as well as coping with their updates in a transparent manner.

## 2. JINI ON THE GRID

The Grid community[1] has very recently payed attention to the Java Jini technology as a viable solution for building a robust Grid system. Jini offers a variety of features necessary for distributed systems, such as discovery of resources, dynamic federations, distributed leasing, and distributed event management, allowing construction of self-healing system that have no single point-of-failure. It serves as an underlying building block for systems and/or applications in a dynamic environment as per the Grid, where network/machine failures/updates occur frequently. We first give a very brief overview of Jini, pointing out its advantages as well as its deficiencies when straightforwardly employed for the Grid.


SC2001 November 2001, Denver (c) 2001 ACM 1-58113-293-X/01/0011 $5.00

## 2.1 Jini Overview

Jini[5, 9, 16, 17, 8] is a framework for building scalable, robust, distributed systems using Java. It consists of a set of specifications describing the model of operation of a Jini network, including related protocols, classes, interfaces, helper utilities and services. A Jini network is a network of many services. Applications are created by dynamically combining these services in a grouping called the "federation".

A Jini lookup service is the key component providing the useful and unique features of Jini. The primary role of a lookup service is service registration and bootstrapping assistant. It accepts registrations from services, each of which is called a service item, composed of service's proxy object, service ID, and a set of attributes associated with the proxy. A lookup service guarantees the uniqueness of the service among multiple lookup services with an assignment of a service ID, which is a random 128 bit number generated from time at instigation together with the network host address. The lookup service supports a template search based on any combination of the three criteria: service ID, type (Java interfaces or classes) that a service supports, and associated attributes. To search for a service, the client fills out a template using only the fields it is interested in.

A Jini lookup service bootstraps a Jini client and services by delivering a proxy corresponding to the service through the Java dynamic class loading mechanism via the network. Any Java class information that is required to re-constitute the proxy will be dynamically loaded over the network through the codebase annotation mechanism. This is the chief mechanism that gives Jini its power; the client initially does not require any fragments of client software, freeing users from updating or installing it. Moreover, a lease is negotiated between a service and a lookup service following the successful registration of the service item. If this lease is not renewed by the service before it expires, the lookup service removes the service item from its database. This means that any service failing to renew its lease, because of a broken network connection or system failure, won't be offered to requesting clients, bringing system robustness.

The Jini security model is consistent with JDK 1.2's fine-grained, permission-based security model. One can create security policy files to restrict the activity of downloaded proxy objects. One can also be selective about the access permissions granted to proxy objects, restricting them according to codebase or code signings.

## 2.2 Deficiencies of Jini Alone as a Computing Portal Platform

Whereas Jini provides a variety of features seemingly useful for the Grid as well as computing portals in general as stated above, there are some key features lacking in Jini when straightforwardly applied as an underlying substrate for computing portals construction. We discuss such "deficiencies" below:

- Scalability
  Since the Jini lookup service is intended to be used in a local area network setting, by itself it is not appropriate for managing a large number of Jini services spread over a Grid environment, as standard Jini assumes a centralized lookup service. A solution (cf.Figure 1) might use chain of lookups amongst the federation of lookup services, where a lookup service itself could be a full-fledged Jini service registered with another lookup service. However, the overhead of traversing down the chain with lookups will cause considerable overhead, as each step in the traversal will require a series of full Jini service lookup and acquisition. This will become critical in a large-scale Grid environment where a large number of services and resources could potentially exist.

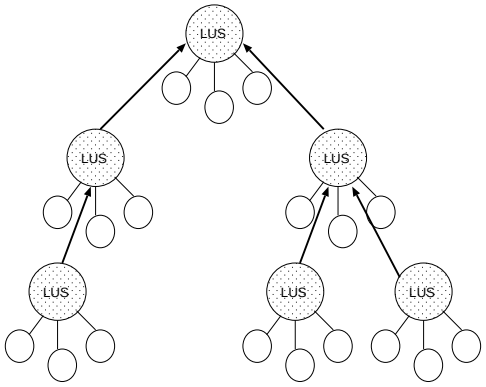


**Figure 1: Hierarchical chains of lookup services**

- Search flexibility
  The Jini lookup service only allows one to search for a service with exact matching. For example, when one wishes to search for a service whose name is "Ninf" but the exact name is vague, it is not possible to search using regular expressions such as "*nin*" or "?inf". Other search strategies are conceivable, but unsupported. A computing portal must support search flexibility up to the extent that standard web portals provide, facilitating various ways to search for a Grid service, matching one's requirements against the characteristics that a particular service provides.

- Tool Support
  In order to make a service Jini-aware, One must perform some amount of programming accompanied with knowledge of Jini. Although such programming can be said to be sufficiently simple, it would nevertheless be preferable if there are tools to help the users make the Grid services Jini-aware in a simpler fashion, and allow their management in a cohesive fashion.

- Security
  Security is one of the concerns explicitly left unaddressed in the Jini specification. It is deemed as implementation dependent with no support for authentication and authorization.

Consequently, although Jini can be used as an underlying infrastructure, we must enhance Jini in order to achieve a more comprehensive support for particularities of the Grid.

# 3. JIPANG SYSTEM—A JINI-BASED PORTALS SYSTEM

In order to exploit Jini technology and cope with its drawbacks in the Grid environment, we propose the JiPANG system, which is a computing portal system and a toolkit. The primary goal of the JiPANG system is to perform uniform higher-level management of the computing resources and services on the Grid, and provide users with a consistent and transparent interface for accessing such services. All entities in the Grid environment including computing resources and services are represented as Jini services. Each entity is registered with a nearby Jini lookup service, so that Jini features such as fault resilience, dynamicity, federation become available.

To overcome the deficiencies, JiPANG builds a layer on top of Jini. For scalability of resource management it leverages the combination of LDAP[4] servers and a collection of Jini lookup services. The Jini lookup services exist at arbitrary locations, managing a collection of registered services, while the LDAP server publishes the information of all the services. For searching and resource discovery it exploits the search capabilities of LDAP, as well as providing a set of tools including the JiPANG browser that allows the user to search and browse through the resources as they become available (or unavailable) to the Grid. JiPANG also facilitates a toolkit that make it easy to adapt existing Grid services as Jini services. We will subsequently give an overview of the architecture of JiPANG, and the details of the individual component.

## 3.1 The JiPANG Architecture

The JiPANG architecture consists of the following components, as illustrated in Figure 2:

- Jini Lookup services
- JiPANG (Grid) Services
- Registration Manager
- Directory services
- Service Broker

### 3.1.1 Jini Lookup Service

Each organization basically manages one or more Jini lookup service(s), and JiPANG Grid services running in the same network are registered with one of them. The latter is the actual component that handles the management of the registered services.

### 3.1.2 JiPANG (Grid) Services

A JiPANG Grid service is a full-fledged Jini service that provides the actual Grid service to the user. By the use of Jini and the higher-level support provided by the JiPANG toolkit, it is basically possible to create all kinds of services for existing Grid systems, tools, and applications. To make an existing service accessible to the Grid, the service provider builds a service proxy object which will be located in the client Java VM and communicating with the service on behalf of the user. The service proxy object will use its own proprietary protocol using sockets directly to communicate with the service logic. It is also registered with a specified Jini lookup service, along with information pertaining to the service such as its location, manufacturer, etc. The JiPANG service toolkit provides a command-line utility to assist this process.

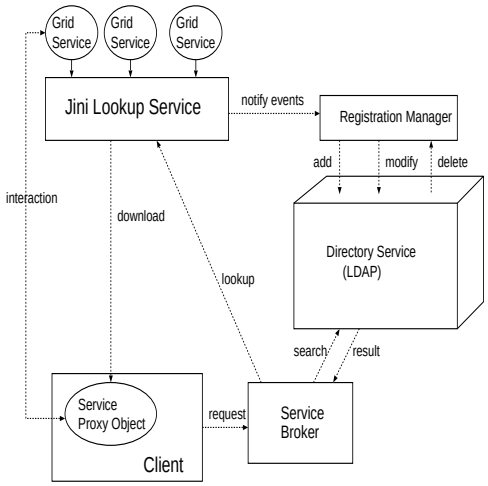


**Figure 2: JiPANG Architecture**

One can also attach a GUI as an attribute of the service, enabling a client to search for and employ the GUI in a portable manner. JiPANG exploits a jini.org project called ServiceUI[7] for this purpose. A service with a GUI can also be executed directly via the JiPANG Browser which allows the user to view all the services registered with the JiPANG system, before the user is even aware of the specification of each service by GUI experimentation.

The service proxy object and its associated GUI service provide a facade(a design pattern) to which some complex service architecture can present a uniform outlook to the user in a portable manner. In fact, by standardizing on an uniform API for a certain class of services, the user need not be concerned at all with the protocol that the proxy object uses, nor the service logic which the service proxy object communicates with. For example, JiPANG allows one to build a uniform interface to GridRPC systems such as Netsolve, Ninf, etc., without the user being aware of which system he is actually using the RPC service of, as we will see in Section 4.

### 3.1.3 Registration Manager

The registration manager is responsible for reflecting the information stored in the Jini lookup service into the LDAP server. This component runs alongside each individual Jini lookup service, waiting for event notifications from the lookup service. The event contains the registration or deletion of services, or modification of some attributes associated with a service item. For example, when the manager obtains the event that a service is registered, it fetches the attributes associated with the service and store them in the LDAP server. Alternatively, if a service is deleted due to lease expiration or service failure, the manager also deletes the entry in the LDAP server.

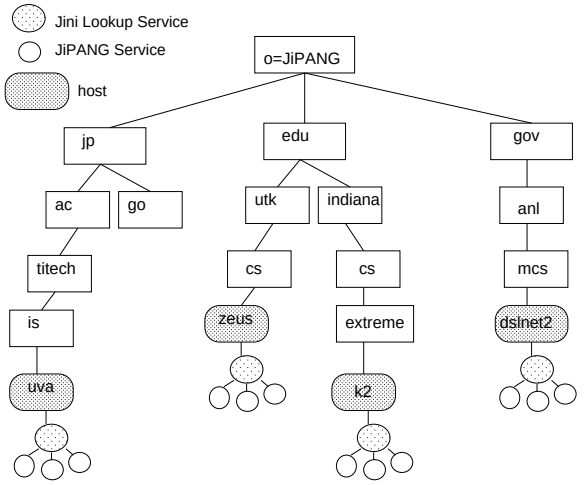


**Figure 3: The JiPANG Directory Service**

### 3.1.4 Directory Service

The directory service is used to publish the location and attributes of all JiPANG services registered with lookup services in the Grid. This component allows the users to search for their desired service in a faster and more flexible manner than directly employing Jini lookup services. Jini lookup services and their associated services are organized as entries in a hierarchical tree called the directory information tree (DIT) (cf.Figure 3). The location of an entry in the DIT is based on organizational structures and other entries it is associated with. For example, a Jini lookup service running at a host named 'uva.is.titech.ac.jp', is located in the directory information tree at: *lookup=uva-lus, dc=uva, dc=is,dc=titech, dc=ac, dc=jp*. The registration of this entry is performed by the registration manager running alongside each Jini lookup service.

The collection of services registered with a lookup service are located at its sibling nodes. As an example, the "Ninf" service running on the node can be located at *service=Ninf, lookup=uva-lus, dc=uva, dc=is, dc=titech, dc=ac, dc=jp*. The set of attributes associated with the service include the service name, proxy class name and its interface name, etc. Its GUI as well as other attributes may be represented as entries as well as described earlier. The attached GUI interface with a service is stored in the sibling node of the service. Likewise, in the case of a service offering a set of computational solvers, such solvers are represented as sibling nodes.

A directory service can contain referral to other directory services that a particular Grid system may maintain for itself. For example, an entry for a particular resource data in the JiPANG directory can refer to an entry in the Globus MDS. This offloads the management of the internal resource management structure that need not be exposed to the client, but otherwise important for the particular Grid system for its own purposes.

### 3.1.5 Service Broker

The purpose of the Service Broker is to select the "best" service, allowing the user to specify only the minimum parameters of interest. The Broker first queries the Directory Service for a set of services that satisfy the client's request. The Broker then filters the set of services according to the criteria in the request. When a service is finally selected out of the subset, the Broker downloads the service proxy object based on the metadata received from the Directory service and deploys it within the client's side of Java VM. The Service Broker can also be extended to serve as alternative resource brokering system for dedicated Grid services such as the GridRPC system.

## 3.2 JiPANG Toolkit

The JiPANG Toolkit provides the user with uniform access to the Grid services, and allows the service provider to create such services in an easy manner. The toolkit consist of three subcomponents, the service toolkit, client toolkit, and the browser.

### 3.2.1 Service Toolkit

The service toolkit allows the Grid service provider to facilitate service registration into the JiPANG system without knowledge of Jini. It consists of several command-line utilities, one of which is called *jipang_register*, allowing the service provider to register the service proxy object with the lookup service as specified in the XML-based configuration file. The configuration file also contains various information regarding the service, and such information are automatically converted into a set of attributes associated with the service. A sample file, which registeres a Ninf services with a host, is shown below.

```
<JipangService>
<lookup host="uva.is.titech.ac.jp" port="4160" />
<proxy> <!-- Proxy class Info -->
<class>org.jipang.grid.ninf.NinfProxy</class>
<param type="java.lang.String">uva.is.titech.ac.jp</param>
<param type="java.lang.String">3030</param>
</proxy>

<serviceInfo>
<name>Ninf</name>
<manufacturer>Hidemoto Nakada</manufacturer>
<vendor>Electrotechnical Laboratory</vendor>
<version>1.0</version>
<model>new</model>
<serialNumber>1.1.1</serialNumber>
</serviceInfo>

<!-- Grid RPC Information -->
<gridRPC>
<server>
 <host>uva.is.titech.ac.jp</host>
 <port>3030</port>
 <resource> ldap://uva.is.titech.ac.jp:2339/cn=uva, dc=is, dc=titech, dc=ac, dc=
 </server>
</gridRPC>

....
....

<!-- Service GUI Information -->
<guiInfo>
<name>NinfBrowser</name>
<description>This is a
main user interface provided by Ninf service
</description>
<role>net.jini.lookup.ui.MainUI</role>
<toolkit>java.awt</toolkit>
<type>Frame</type>
<factory>org.jipang.grid.ninf.NinfUIFact</factory>
</guiInfo>
</JipangService>
```

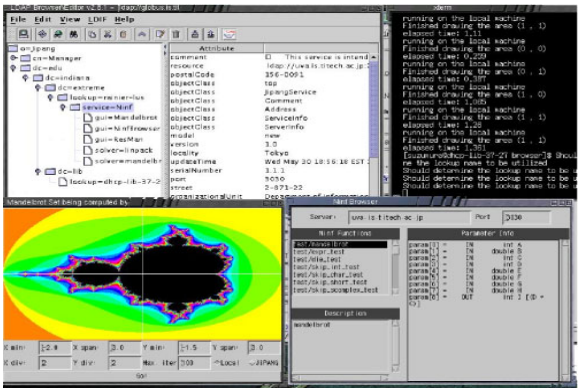

**Figure 4: The JiPANG Browser**

### 3.2.2 *Client Toolkit*

The client toolkit is a set of Java class libraries that allows users access to a collection of services registered within the JiPANG system from his Java program. Alternatively, the service provider can use the toolkit to create a front-end application for a class of Grid services. With this API, a user does not require prior knowledge of the location of the service, nor install any client packages. Here is a sample API provided by this toolkit:

```
public JipangServiceInfo[] searchService(String service, String
                  filter, String baseDN) throws JipangException ;
public Object getProxy(String dn) throws JipangException;
```

The first method takes three arguments: service name, a character for representing filtering condition in the LDAP format, and the base distinguished name; and returns an array of service information that satisfies the search condition. This API encapsulates the process of querying the LDAP server with a set of attributes and retrieve appropriate services that match the criteria. The second method obtains the service proxy object whose distinguished name is registered in the LDAP server from a corresponding Jini lookup service.

### 3.2.3 *JiPANG Browser*

The JiPANG Browser provides a Microsoft Windows Explorer-like interface to the JiPANG system. It allows users to explore a variety of services stored in the Directory Service, searching for a desired service. The browser can be customized so that application-specific browsing and features can be integrated. We have developed the browser based on an existing LDAP Browser which is mainly attributed to Jarek Gawor, Illionis Institute of Technology. It is entirely written in Java with the help of the JFC(SwingSet), JNDI, and Jini class libraries. The key features of the browser are that it itself is a JiPANG Grid service, and also that users can launch services including GUI services directly from the Browser. The figure 4 shows a screenshot of the browser in action, with GUI services listed in the browser.

## 4. EXAMPLE JIPANG GRID SERVICES

Whereas the JiPANG Service Toolkit allows one to build arbitrary Grid services, we show prototype example Grid services we have built using JiPANG. They include the GridRPC service that abstracts out the differences between a class of GridRPC systems such as Netsolve and Ninf, the Instrumentation Service built using NWS, and Globus services that provide full fledged Globus services through the CoG toolkit in a portable manner.

### 4.1 GridRPC Service

GridRPC is a middleware that provides remote library access and task-parallel programming model on the Grid. Representative systems include NetSolve, Ninf, etc. The JiPANG service implementing the *GridRPCInterface* interface defined below deals with the interaction with remote computational servers in a portable, transparent manner, by defining a uniform GridRPC invocation method called *jipangCall(String problem)*. A *jipangCall* invocation results in problem submission and retrieval of the result from the server, irrespective of whether it is a Netsolve server, Ninf server, CORBA server, or an unknown server to be defined in the future. The user simply writes one application using *jipangCall*, and the latest version of the proxy object, would be downloaded on-the-fly in the client's Java VM, and speaks the particular protocol as required by the RPC server. To date, we have developed GridRPC services implementing this interface, that can call a NetSolve server, a Ninf server, or a simple compute service which calls JLAPACK[11] routines in a remote server via Java RMI.

```
public interface GridRPCInterface
                    extends JipangInterface {
  public void setArg(Vector vec) throws JipangException;
  public JipangExecInfo jipangCall(String func)
                                  throws JipangException;
}
```

Here is an example call being made with *jipangCall*:

```
public class GridRPCTest{
  public static void main(String[] args) {
  ....
  /** instance a jipang client */
  GridRPCClient client = new GridRPCClient();

  /** set arguments to be passed to a GridRPC server */
  client.setArg(max);
  client.setArg(mag);
  client.setArg(xsize);
  client.setArg(ysize);
  client.setArg(left);
  client.setArg(right);
  client.setArg(top);
  client.setArg(bottom);
  client.setArg(output);

  /** submit a problem and retrive the result */
  JipangExecInfo execInfo = client.jipangCall("mandelbrot");
  System.out.println(execInfo.toString());
  ...
}
```

### 4.2 Instrumentation (NWS) Service

The instrumentation service we have developed is a proxy to the Network Weather Service (NWS)[23], and provides accurate forecasts of dynamically changing performance characteristics from a distributed set of metacomputing resources. By specifying in the query, an appropriate NWS instrumentation data as well as forecasts can be retrieved, irrespective of the client location. Also, other instrumentation services

can be utilized with the same set of APIs as is done for GridRPC (although some features such as forecasts may not be available, or only return the last instrumented data.)

```
public interface NWSInterface extends JipangInterface {
   public String getNameServer(String nameWithPort)
                            throws JipangException;
   ...
   public String[] getForecasts(String sourceMachine,
        String destinationMachine,String experimentName,
        int atMost) throws JipangException;
   public String[] getMeasurements(String sourceMachine,
       String destinationMachine,
       String experimentName,int count, double sinceWhen)
                  throws JipangException;
}
```

## 4.3 Globus Service

The Java Commodity Grid Toolkit(CoG Kit) for Globus provides a rich set of classes for Java programmers to access basic Globus services. By using CoG we have built a Jini service that publishes most of Globus services. An advantage of using Globus services in this style is that, it allows construction of thin-client software, for which Globus services can be dynamically installed or updated as well as loaded at the time of use. As such, as long as Globus is registered and maintained on at least one machine readily accessible and published to the JiPANG directory, it becomes accessible, with latest versions of the CoG/Globus automatically downloaded/used as long as they are properly maintained at the service site, freeing the client users from maintaining Globus himself. We currently have defined a set of APIs to provide interfaces to the low-level Globus services and application interfaces such as RSL, GRAM, MDS, GASS, GSI, GSI-FTP, and GARA. The APIs are designed to resemble the original ones for user familiarity. Below are an example GRAM interface and a sample program using the interface.

```
public interface GlobusGramInterface extends JipangInterface {
   public void ping(String resourceManagerContact)
                            throws JipangException;
   public void request(String resourceManagerContact,
                       String rsl)
                            throws JipangException;
   ....
}
```

```
public class GramTest {
 public static void main(String[] args){

  GlobalGramInterface proxy;

  try{
    JipangClient client
      = new JipangClient("globus.is.titech.ac.jp", "389");
    proxy = (GlobusGramInterface)client.getProxy(dn);
  } catch(JipangException ex){
    ex.printStackTrace();
    System.exit(1);
  }

  String dn = "service=Gram,lookup=matsulab-lus,
               dc=is,dc=titech,dc=ac,dc=jp";
  String rsl =
   "&(executable=/bin/sleep)(directory=/tmp)(arguments=15)";

  JipangGramJob job = new JipangGramJob(rsl);
  String contact = null;

  if (args.length == 0) {
   System.err.println("Usage: java GramTest [resource manager]");
   System.exit(1);
  }

  contact = args[0];
  try {
   job.addListener( new JipangGramJobListener() {
     public void statusChanged(JipangGramJob gjob) {
      System.out.println("Job status change \n" +
       "    ID     : "+ gjob.getIDAsString() + "\n" +
 "    Status : "+ gjob.getStatusAsString());
     }
   });

  System.out.println("submitting job1...");

  /*
   * call a method on the downloaded proxy
   */
  proxy.gramRequest(contact, job);
  System.out.println("job submited: " + job.getIDAsString());
  try {
      while ( proxy.gramGetActiveJobs() != 0 ) {
  Thread.sleep(2000);
   } catch(Exception e) {
  System.out.println(e.getMessage());
   }
}
```

# 5. DISCUSSION

We have already covered the properties of JiPANG with respect to the deficiencies we have mentioned earlier. We now briefly discuss whether JiPANG satisfies other requirements as a computing portals system as depicted in Section 1:

- High Usability and Portability

  In JiPANG, usability is achieved by the fact that the service proxy object is downloaded on-the-fly at runtime onto the client Java VM, allowing the user to interact transparently with the latest version of the service, freeing the users from tedious installation, maintenance, and update tasks. Portability is achieved in that a Grid service is provided over the network via a proxy Java object, runnable on any device which supports Java VM.

- Interoperability and Dynamicity

  JiPANG provides users with one consistent Java library toolkit to access a wide-range of Grid services. the JiPANG toolkit facilitates the development of a Grid application combining different kinds of Grid services. It also allows more dynamic component programming, coupling various services together as each service will be downloaded on-the-fly as required. As an example, consider an application that consists of three phases, first phase transferring a large amount of data from a remote database server to a computing server as input, second phase computing with various solvers, and finally visualizing the ouput obtained from the compute server. With JiPANG, the application can firstly utilize Globus GASS service for file transfer, the Ninf service to invoke multiple solvers in the computing phase, and finally invoke the general visualization services for visualizing the ouput.

- Uniformity

  In JiPANG uniformity is achieved by implementing the common Java interface for a class of services providing

similar functionalities. As we have seen, we have defined a common interface called GridRPCInterface for various Grid RPC systems. Users can access such services implementing this interface from the same client source code without any modifications.

## 6. RELATED WORK

Webflow[12] is a pioneering computing portals work where end-user interfaces are provided using standard web browsers, and a group of http servers are used not only as web servers but also compute server proxies using CGI technology. As such the client characteristics are limited to those of Web/CGI, e.g., one will have to handcraft scripts to interface existing Grid services as CGI services. GridPort[18] is a collection of services, scripts and tools that allow developers to connect Web-based interfaces with the computational Grid behind the scenes, and are based on both Grid technologies such as Globus and standard Web technologies such as CGI and Perl. GridPort facilitates numerous features, but does not provide transparent, uniform interface in a streamlined architecture as is with JiPANG. The CoG[15] project has undertaken the design and development of a set of Commodity Grid Toolkits (CoG Kits), that define and implement a set of general components that map Globus functionality into a commodity environment such as Java, CORBA, DCOM, etc. CoG and JiPANG are somewhat complimentary because one of CoG's primary objective is to define a common API as a specification, whereas JiPANG's approach is to allow incorporation of services using Jini technology.

## 7. FUTURE WORK

This paper has given an overview of the JiPANG system, a computing portals system and toolkit based on Jini. The resulting infrastructure will allow integration of a large number of future Grid services on the network accessible in a portable, uniform way by Grid application developers.

There are numerous work to be done. Firstly, enhanced development and documentation work is necessary to make more Grid services available to the client users. Some of the Globus services may be too low level as a client service, and we need to build more layers on top to allow easier application development. We need to experience of application-level services as is with the Punch system[19]. Security is one aspect that is not well covered neither with Jini nor JiPANG. We need to integrate the standardization efforts of the Grid security model into JiPANG.

## 8. ACKNOWLEDGMENTS

The authors would like to sincerely thank Jack Dongarra, Rich Wolski, and the NetSolve team in University of Tennessee, Dennis Gannon and his group in Indiana University for providing us with their applications and giving us their ideas and comments. We also thank to the whole CoG team for providing the CoG toolkit. In particular, we thank Jarek Gawor for allowing us to extend and use his Java-based LDAP browser.

## 9. REFERENCES

[1] Global Grid Forum Jini Working Group. http://www.mcs.anl.gov/gridforum/jini/.

[2] Grid Computing Environments Working Group. http://www.computingportals.org/.

[3] GridRPC Tutorial. http://ninf.etl.go.jp/papers/gridrpc_tutorial/.

[4] OpenLDAP Project. http://www.openldap.org.

[5] The Jini Community. http://www.jini.org/.

[6] The Neos Project. http://www-neos.mcs.anl.gov/.

[7] The ServiceUI Project. http://artima.com/jini/serviceui/index.html.

[8] P. M. Ahmed Al-Theneyan and M. Zubair. Enhancing Jini for use across non-multicastable networks. In *ICASE Report No.2000-34*, 2000.

[9] K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. The jini specification, 1999.

[10] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. In *Proceedings of Super Computing '96*, 1996. http://www.cs.utk.edu/netsolve/.

[11] J. D. David M.Doolin and K. Seymour. JLAPACK -Compiling LAPACK Fortran to Java. June 1988.

[12] W. F. Erol Akarsn, Geoffrey C.Fox and T. Haupt. Webflow - high-level programming environment and visual authoring toolkit for high performance distributed computing. In *Proceedings of Supercomputing '98*, 1998. http://www.npac.syr.edu/users/haupt/WebFlow/.

[13] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S.Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proc. 6th IEEE Symp. on High-Performance Distributed Computing*, pages 365–375, 1997.

[14] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proc. IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[15] J. G. Gregor von Laszewski, Ian Foster. Cog kits: A bridge between commodity distributed computing and high-performance grids,a java commodity grid kit. In *ACM 2000 Java Grande Conference*, June 2000.

[16] A. W. Jim Waldo, Geoff Wyant and S. Kendall. A Note on Distributed Computing. In *Sun Microsystems Technical Reports*, November 1994.

[17] S. Li. *Professional Jini*. Wrox Press Ltd.n, 29 S.LA SALLE ST, SUITE 520 CHICAGO IL 60603, 2000.

[18] S. M. Mary Thomas and J. Boisseau. Development of web toolkits for computational science portals: The npaci hotpage. In *Proceedings of HPDC 9*, pages 308–309, August 2000.

[19] R. J. F. Nirav H. Kapadia and J. A. B. Fortes. Punch: Web portal for running tools. In *IEEE Micro.*, May-June 2000.

[20] S. Sekiguchi, M. Sato, H. Nakada, and U. Nagashima. – Ninf – : Network base information library for globally high performance computing. In *Proceedings of Parallel Object-Oriented Methods and Applications (POOMA)*, Feb. 1996. http://ninf.etl.go.jp/.

[21] T. Suzumura, T. Nakagawa, S. Matsuoka, H. Nakada, and S. Sekiguchi. Are Global Computing Systems Useful ? Comparison of Client-server Global Computing Systems Ninf, NetSolve versus CORBA. In *Proceedings of the 14th International Parallel and Distributed Processing Symposium(IPDPS '00)*, 2000.

[22] e. Warren Smith. An Evaluation of Alternative Designs for a Grid Information Service. In *HPDC 2000 Proceeding*, pages 185–192. GlobusTeX Users Group, March 2000.

[23] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.