Mobile Access to Grid Infrastructure

Technical Report

Draft 2.1

Authors:

Imran Ahmad Rao

Umar Kalim

Ali Hassan

Hassan Jameel

{imran, umar, ali, hassan}@oslab.khu.ac.kr

November 2005

Table of contents

Executive summary

Currently, access to Grid services is limited to resourceful devices such as desktop PCs but most mobile devices (with wireless network connections) cannot access the Grid network directly because of their resource limitations. Yet, extending the potential of the Grid to a wider audience promises increase in flexible usage and productivity. In this technical report we present a middleware architecture that addresses the issues of job delegation to a Grid service, support for offline processing, secure communication, interaction with heterogeneous mobile devices and presentation of results formatted in accordance with the device specification. This is achieved by outsourcing the resource intensive tasks from the mobile device to the middleware. We also demonstrate through formal modeling using Petri nets that the addition of such a middleware causes minimum overhead and the benefits obtained outweigh this overhead.

Chapter 1: Introduction to Mobile Grid Middleware

1.1 Overview

Grid [18] computing permits participating entities connected via networks to dynamically share their resources. Extending this potential of the Grid to a wider audience, promises increase in flexibility and productivity, particularly for the users of mobile devices who are the prospective consumers of this technology.

Consider a teacher who wants to augment his lecture with a heavy simulation test. He uses his PDA to access a Grid service and submit the request. The service after executing the request compiles the results which are then distributed to the mobile devices of the registered students of that course. Similarly a doctor on the way to see his patient, requests a Grid medical service to analyze the MRI or CT scans of the patient from his mobile device. By the time he meets his patient; the results would be compiled and presented on his mobile device to facilitate the treatment.

The clients that interact with the Grid middleware to accomplish a task are required to use client end libraries. These libraries are relatively resource intensive considering the limitations of mobile devices. Conceiving a distributed system that uses these libraries directly will not be a practical mobile system because of the resource demands.

Moreover, most of the conventional distributed applications are developed with the assumption that the end-systems possess sufficient resources for the task at hand and the communication infrastructure is reliable. For the same reason, the middleware technologies for such distributed systems usually deal with issues such as heterogeneity and distribution (hence allowing the developer to focus his efforts on the functionality rather than the distribution).

1.2 Problem Definition

The issues that primarily affect the design of a middleware for mobile systems are: *mobile devices*, *network connection*, and *mobility*. Firstly, due to the tremendous progress in development of mobile devices, a wide variety of devices are available

which vary from one to another in terms of resource availability. On one hand, laptops offer relatively powerful CPUs and sufficient primary and secondary storage. On the other hand devices like cell phones have scare resources and supplementing these resources is either expensive or impossible altogether. Secondly, in mobile systems, network connections generally have limited bandwidth, high error rate and frequent disconnections due to power limitations, available communication spectrum and user mobility. Lastly, mobile clients usually interact with various networks, services, and security policies as they move from one place to another.

Considering the assumptions and characteristics of conventional middleware technologies it is quite evident that they are not designed to support mobile systems adequately. Instead, they aim at a static execution platform (where the host location is fixed) and the network bandwidth does not vary. Hence, given the highly variable computing environment of mobile systems, it is mandatory that modern middleware systems are designed that can support the requirements of mobile systems such as *dynamic reconfiguration* and *asynchronous communication*. As the environment of a mobile device changes, the application behaviour needs to be adjusted to adapt itself to the environment. Hence dynamic reconfiguration is an important building block of an adaptive system. Note that middleware that provides the facility of dynamic reconfiguration needs to identify the changes in the environment and consequently inform the application to adapt itself or initiate reallocation of resources accordingly. Also, the interaction approach between the mobile client and the host dictates the effectiveness and efficiency of a mobile system. Asynchronous interaction deals with problems of high latency and disconnected operations that may arise with other interaction models. A client using asynchronous communication can issue a request and continue with its local operations and collect the output later. Hence the client and server modules are not required to execute concurrently to communicate with each other. Such an interaction permits reduction in bandwidth usage, decouples the client and server modules and improves the scalability of a system.

1.3 Organization of the Paper

In this paper:

- We present an architecture for a middleware (Section 2) enabling heterogeneous mobile devices access to Grid services and implement an application toolkit that acts as a gateway to the Grid. This middleware provides support for delegation of jobs to the Grid, secure communication between the client and the Grid, offline processing, adaptation to network connectivity issues and presentation of results in a form that is in keeping with the resources available at the client device.
- Next four sections explain three main components of our framework. Section 3, explain the adaptive middleware. Section 4 is about autonomous knowledge mangment and section 5 is about security module. Section 6 explains the interaction of middleware with Grid.

- We demonstrate (Section 7) compares our work with existing efforts and we discuss how addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead.

- Last section, namely Section 8, gives summary of our work and describes the future work.

Chapter 2: MAGi Architecture

2.1 Architecture details

The middleware service is exposed as a web service to the client applications. The components of the middleware service (as shown in Figure 1) are discussed succinctly as follows:

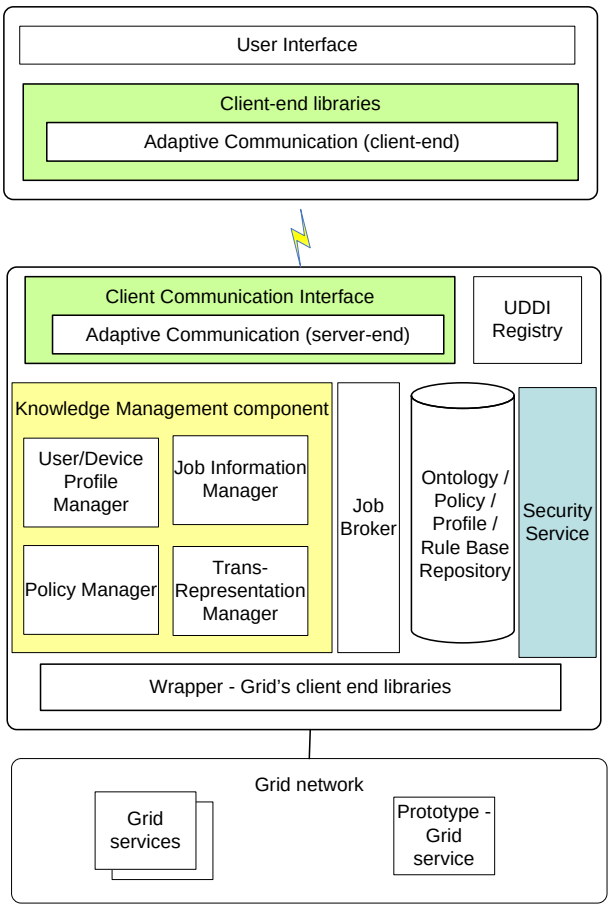

Figure 2.1. Architecture

Chapter 3: Application Aware Adaptation

The software systems for hand-held devices operating in wireless environments require adaptation to the variations in the environment (such as fluctuating bandwidth). This translates to maintenance of service availability in preferably all circumstances. Particularly when considering Quality of Service, maintaining service availability for hand-held devices in the face of varying network conditions and disconnections is of utmost importance. In this research endeavour we propose that systems operating on hand-held devices and in wireless environments should be based on the mechanisms of reflection and code mobility in order to be able to adapt to the varying network conditions. This not only allows the application to continue executing in varying circumstances, but also in entirely disconnected modes.

We present a framework and discuss the implementation as well as the evaluation of the infrastructure that allows developers to design disconnection-aware applications which automatically redeploy essential components to appropriate locations to maintain service availability

3.1 Discovery service

The discovery of the middleware by mobile devices is managed by employing a UDDI registry [6], [7]. Once the middleware service is deployed and registered, other applications/devices would be able to discover and invoke it using the API in the UDDI specification [7] which is defined in XML, wrapped in a SOAP envelop and sent over HTTP.

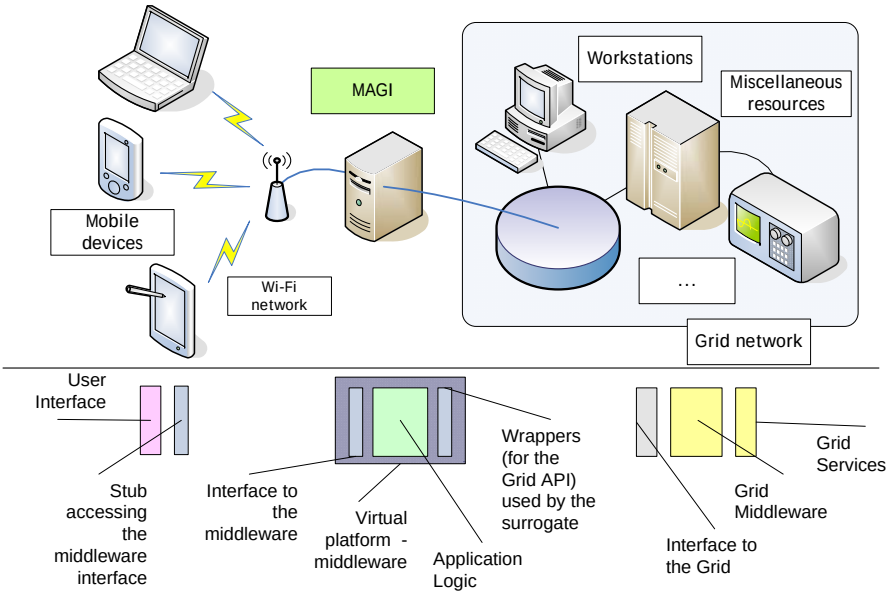

Figure 3.1. Deployment model and the architecture

3.1.1 How does it work?

For the client the first step is establishing a connection with the registry. However this is possible only if we have access to the registry. Since we are using the Java WSDP Registry Server [19] for the prototype version, we configure it accordingly and in order to access the registry and execute queries we create a JAXR client for the mobile device.

A client **creates a connection** from a connection factory. A JAXR provider may supply one or more preconfigured connection factories that clients can obtain by looking them up using the Java Naming and Directory Interface (JNDI) API.

At the moment JAXR does not supply preconfigured connection factories. Instead, a client creates an instance of the abstract class `ConnectionFactory`:

```
import javax.xml.registry.*;
...
ConnectionFactory connFactory =
    ConnectionFactory.newInstance();
```

To create a connection, the client first creates a set of properties that specify the URL or URLs of the registry or registries being accessed. For example, the following

code provides the URLs of the query service and publishing service for the test registry. (There should be no line break in the strings.)

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
    "http://uddi.ibm.com/testregistry/inquiryapi");
props.setProperty("javax.xml.registry.lifeCycleManagerURL
",
    "https://
uddi.ibm.com/testregistry/protect/publishapi");
```

With the Java WSDP implementation of JAXR, if the client is accessing a registry that is outside a firewall, it must also specify proxy host and port information for the network on which it is running. For queries it may need to specify only the HTTP proxy host and port; for updates it must specify the HTTPS proxy host and port.

```
props.setProperty("com.sun.xml.registry.http.proxyHost",
    "myhost.mydomain");
props.setProperty("com.sun.xml.registry.http.proxyPort",
    "8080");
props.setProperty("com.sun.xml.registry.https.proxyHost",
    "myhost.mydomain");
props.setProperty("com.sun.xml.registry.https.proxyPort",
    "8080");
```

The client then sets the properties for the connection factory and creates the connection:

```
connFactory.setProperties(props);
Connection connection = connFactory.createConnection();
```

The `makeConnection` method in the code (`JAXRQuery.java`) shows the steps used to create a JAXR connection.

The implementation of JAXR in the Java WSDP allows you to **set a number of properties** on a JAXR connection. Some of these are standard properties defined [20] in the JAXR specification. Other properties are specific to the implementation of JAXR in the Java WSDP.

- Most of these properties must be set in a JAXR client program. For example:

```
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
  "http://uddi.ibm.com/testregistry/inquiryapi");

props.setProperty("javax.xml.registry.lifeCycleManagerURL
",
```

```
     "https://
uddi.ibm.com/testregistry/protect/publishapi");
   ConnectionFactory factory = ConnectionFac-
tory.newInstance();
   factory.setProperties(props);
   connection = factory.createConnection();
```

- The `postalAddressScheme`, `useCache`, and `useSOAP` properties may be set in a `<sysproperty>` tag in a `build.xml` file for the `Ant` tool. For example:

```
   <sysproperty key="useSOAP" value="true"/>
```

These properties may also be set with the `-D` option on the `java` command line.

After creating the connection, the client uses the connection to **obtain a `RegistryService` object** and then the interface or interfaces it will use:

```
RegistryService rs = connection.getRegistryService();
BusinessQueryManager bqm = rs.getBusinessQueryManager();
BusinessLifeCycleManager blcm =
    rs.getBusinessLifeCycleManager();
```

Typically, a client obtains both a `BusinessQueryManager` object and a `BusinessLifeCycleManager` object from the `RegistryService` object. If it is using the registry for simple queries only, it may need to obtain only a `BusinessQueryManager` object.

The simplest way for a client to **use a registry** is to query it for information about the organizations that have submitted data to it. The `BusinessQueryManager` interface supports a number of find methods that allow clients to search for data using the JAXR information model. Many of these methods return a `BulkResponse` (a collection of objects) that meets a set of criteria specified in the method arguments. The most useful of these methods are:

- `findOrganizations`, which returns a list of organizations that meet the specified criteria--often a name pattern or a classification within a classification scheme
- `findServices`, which returns a set of services offered by a specified organization
- `findServiceBindings`, which returns the service bindings (information about how to access the service) that are supported by a specified service

The `JAXRQuery` program illustrates how to query a registry by organization name and display the data returned. The `JAXRQueryByNAICSClassification`

and `JAXRQueryByWSDLClassification` programs illustrate how to query a registry using classifications.

3.2 Communication Interface with the Client Application

The interface advertised to the client application is the communication layer between the mobile device and the middleware. This layer enables the middleware to operate as a web service and communicate via the SOAP framework [8].

3.2.1 Adaptation to disconnected operations

The advertisement of the mobile-to-Grid middleware as a web service permits the development of the architecture in a manner that does not make it mandatory for the client application to remain connected to the middleware at all times while the request is being served.

We focus on providing software support for offline processing at the client device. We propose that a mobile computing system for handheld devices must be based on the combination of reflection [21] and code mobility [22]. Reflection is a fundamental technique that supports both introspection and adaptation. In order to maintain service availability in a distributed system (i.e. offline processing), in case of varying circumstances (such as fluctuating bandwidth or disconnection), the middleware can utilize this mechanism along with code mobility to automatically redeploy essential components to appropriate locations. This way, the services that need to be executed at a platform residing in a portion of the network reachable only through an unreliable link (at that particular instance) could be relocated temporarily and hence maintain the desired level of Quality of Service by maintaining service availability.

To introduce the capability of dynamic reconfiguration to achieve the above mentioned objectives, the system must posses certain characteristics, such as, it should be based on a distributed object framework, the system must be able to redeploy/replace components, it should be able to recover gracefully in the case of faults etc. We employ Java [23] as the technology to validate our hypothesis by providing the following mechanisms.

> **Code mobility** Java allows for code mobility and portability as the runtime environment (JVM) is an interpreter. The compiled code is maintained as *byte code* and converted to machine code by interpretation at runtime. Java also provides an object; *class loader* which can dynamically load classes at runtime from a variety of sources such as disk or network. The class loader is invoked by the runtime environment to translate a class name into the class

reference. This is done by loading the class (if it is not loaded already) and instantiating a reference.

Dynamic binding An essential element for code mobility is dynamic binding. This allows the system to determine what code to execute, at runtime. Java provides this support by postponing the binding of code until an invocation has been made, after which the class may be loaded and executed.

Serialization and Deserialization A crucial aspect of code mobility is to transfer the execution state along with the code. Java provides support for this via *object serialization*. This allows the translation of a graph of objects to a stream of bytes which may be sent over a network. Each object to be serialized is required to implement the *Serializable* interface. Java also provides the facility to customize the process of serialization. Similarly, the reverse process of conversion from the stream of bytes to an object graph (deserialization) is also supported.

However, the ability of the components in a system to redeploy themselves is not sufficient. There is a requirement for appropriate policies to determine when, how and which components to redeploy. In this regard, we suggest that the application developer be involved in the process of implementing an adaptive system, hence the name application aware adaptation. The middleware provides the means for adaptation, whereas the developer decides when to use which facility. This would not only allow different application domains to use the middleware, but would also result in one of the most optimized implementations as the domain expert would be involved in the process of adaptation.

3.2.1.1 Application Aware Adaptation

In order to narrow down the scope of the problem, we distinguish between voluntary and involuntary disconnection. The former refers to a user-initiated event that enables the system to prepare for disconnection, and the latter to an unplanned disconnection (e.g., due to network failure). With involuntary disconnection the system needs to detect the disconnection (and reconnection), requires to be pessimistically prepared for it at any moment and pro-actively prepare a response. In order to predict a disconnection various efficient and accurate techniques [24] have been developed which may be employed. Here we are only focusing on voluntary disconnections and (for the time being) consider the prediction sub-system as a black-box which employs prediction algorithms as proposed by [24]. However note that whether the disconnection is voluntary or involuntary, the steps for the remedy will always be the same.

The fundamental notion is that under normal circumstances, the system would continue to operate according to the remote evaluation paradigm. However if the network bandwidth is predicted to drop below a certain threshold, the framework must allow the system to adapt to this change (before it happens). The adaptation mechanism could facilitate the relocation of certain objects (with their state) to the clients platform in order to allow the client to continue his operations. Relocation of these ob-

jects is subject to the policy defined by the application developer while considering the classification discussed in the following sections. These objects may posses the same or partial functionality as compared to the implementation present at the server end. Once the network bandwidth becomes available, the objects will be moved back to the server (along with their state) to resume normal operations. The relocation from the client to the server is also subject to the policy exercised by the classification of objects, discussed in the following sections.

3.2.1.1.1 Floating objects and their characteristics

The objects dealing with the process of adaptation to disconnected operations must posses certain characteristics in order to be considered disconnection-aware. We refer to such objects as *floating objects*. These floating objects are the corner-stone for the process of adaptation whenever a change in the network environment occurs.

DisconnectionManagement Interface Each floating object must implement the DisconnectionManagement interface. The DisconnectionManagement interface advertises two primary methods; *disconnect* and *reconnect*. These methods are invoked by the framework upon disconnection and reconnection respectively. The use of *disconnect* is to compile the object state and later transfer it in marshalled form over the network. Similarly *reconnect* is used to perform the process of reconciliation among the objects upon reconnection. Details of the working are explained in the section *Disconnection and Reconnection Management.*

Implementation for local execution The classes intended to be used locally may advertise their interface in the form of *Application Logic*. The instances of the implementation classes would be disconnection aware and would implement the application logic. However this implementation may be partial unlike the remote implementations. This is because these implementations will be used in the face of a disconnection, when the resources at the server are unavailable.

Implementation for remote execution This interface would extend the *Application Logic* interface. Under normal circumstances the implementation classes for these interfaces *Remote Application Logic* would be used remotely. They would extend the interface of their counterpart implementations; the class for local implementation, and would also be disconnection aware. These implementations would be fully functional in all aspects.

As the primary components required to participate in the process of adaptation are supposed to posses these characteristics, from programming point of view, the developers are forced to identify which components are disconnection-aware and thus are the only ones that need to be notified in case of a disconnection.

3.2.1.1.2 Classification of the references of objects

Here we provide a set of possible reference types for the components classified as disconnection aware. This classification has been inspired by [25]. Some of these types apply only to disconnection, some only to reconnection, and some to both. This set of reference types is by no means intended to be complete. We intend to add other classifications as practical need for them arises. Here we focus on the semantics and defer technical aspects of the implementation of these references to section *Disconnection and Reconnection Management.*

Classification w.r.t. disconnection There are three types of disconnection-aware references that determine the behaviour of referencing and the referenced components upon disconnection. This classification is with respect to the transfer of components, their state and the management of references at the disconnecting client.

> **Log** After disconnection, the invocations that are made on this reference are logged locally. These invocations are then executed upon reconnection. Such references pose a constraint that all invocations must not require return parameters, in order to ensure non-blocking invocations.
>
> **Substitute** This type suggests that upon disconnection a temporary but local reference should be used to replace the remote reference. The local reference must have the same interface as the remote reference, however it may have a different (or limited) functionality. Such references will be used, primarily when the remote references cannot be migrated to the client for local execution (as such references might be dependent on non-transferable resources such as databases).
>
> **Replica** This type suggests that the reference should be replicated and transferred (with its state, as per the policy) to the client (before the disconnection) for local execution to maintain service availability. This local reference is then used in the same manner as the remote reference was used. On the other hand the original reference continues to exist at the server end. The replicated reference possesses the same state as the original reference, before the disconnection. Also these references are transitive in nature. If they comprise of replica references, they are also duplicated and transferred (recursively) in the same manner. Using this classification the designer can determine which components are essential for the application and hence must be replicated.

Classification w.r.t. reconnection There are four types of disconnection-aware references that determine the behaviour of referencing and the referenced components upon reconnection, with respect to transfer of components, their state and the management of references at the reconnecting client.

Latest This type suggests that while comparing the original reference at the server end and the temporary reference at the client end, the state of the reference with latest time-stamp is maintained while the other is discarded. Such types pose a constraint that the clocks of the client and the server must be synchronized.

Revoke This type suggests that the entire changes made on the replicated or substituted reference are dropped in favour of the state of the original reference. The notion is similar to changing the state of a variable, passed by value. The reference is used to make invocations, however the changes in the state of the concerned reference are discarded.

Prime Prime references are contrary to revoke references. The state of the original reference is discarded in favour of the Prime reference. Thus effectively overwriting the original reference.

Merge Neither the replicated/substituted reference nor the original reference is clearly identified to be maintained and there is a need for conflict detection and resolution techniques which are elaborated below.

Conflict resolution while merging Without the knowledge of the application domain, it is impractical to suggest a mechanism which resolves conflicts among the components being merged. Therefore, continuing with the notion of application-aware solutions we suggest the use of callback methods. All disconnection-aware references may implement the Merge Interface. This interface advertises a merge callback method that the application developer must implement. This merge method would be invoked on the replicated reference and would accept the original reference as a parameter. Whenever a reconnection occurs (or whenever required) the system would invoke the merge method for reconciliation.

3.2.1.1.3 Disconnection/Reconnection management

Generally, in a distributed object system, an object interface specification is used to generate a server implementation of a class of objects, an interface between the object implementation and the object manager (called an object skeleton) and a client interface for the class of objects (called an object stub). The skeleton is used by the server to create new instances of the class of objects and to route remote method calls to the object implementation. The stub is used by the client to route transactions (method invocations, mostly) to the object on the server. On the server side, the class implementation is passed through a registration service, which registers the new class with a naming service and an object manager, and then stores the class in the server's storage for object skeletons. However when it comes to maintaining service availability in the face of a disconnection, there is a need to relocate the required server code

(with partial or complete functionality) to the client, in order to make local processing possible.

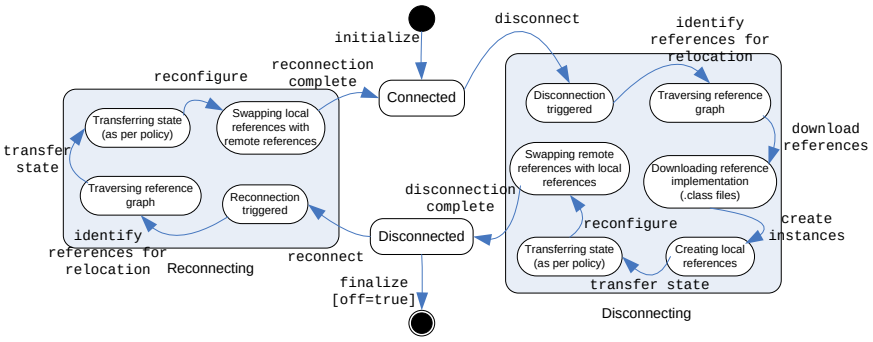

Figure 3.2. State-transition diagram

3.2.1.1.4 Working

The state-transition diagram in the figure 3.2 summarizes the working of the framework. Once a disconnection event is fired, the framework propagates the event to all disconnection-aware references by invoking the *disconnect* method. This method prepares the reference for the disconnection. Once the implementation class of the object is transferred, using Java Serialization techniques the object state is transferred to the client's platform. The local implementation class is then instantiated appropriately and the object state is assigned to the new (local) instance, which possesses the same interface. It may be noted that the framework maintains a sufficient state of each reference in order to restore the system to the state before disconnection. From here on the client application utilizes the local functionality instead of accessing the server implementation.

Similarly, when the network bandwidth is available again, the framework fires the reconnection event and invokes the *reconnect* method. The state of the object is prepared and transferred via Java Serialization methods. The process of reconciliation is done at the server with reference to the reconnection type of the reference, as discussed earlier.

It may be noted that the behaviour of the reference after disconnection and reconnection is subject to its classification as discussed earlier.

> **Pre-installation** The performance of the framework may be drastically improved if the implementation of the objects that are to be relocated is already present at the client. If so, the transfer of state will be the only major obstacle before a disconnection occurs.

Once a disconnection is predicted by the framework, intuitively, the available bandwidth will not be sufficient to relocate the code as well as the state. It may be noted that in such circumstances the bandwidth would be continuously dropping. Thus if the object state is the only data to be relocated, the systems efficiency would increase dramatically. Therefore we are more inclined towards the notion that the implementation code of the objects which might be relocated should preferably be present with the client. It may be noted that these implementations may be partial or the same as the remote implementations. Each object may then be instantiated as per the requirements.

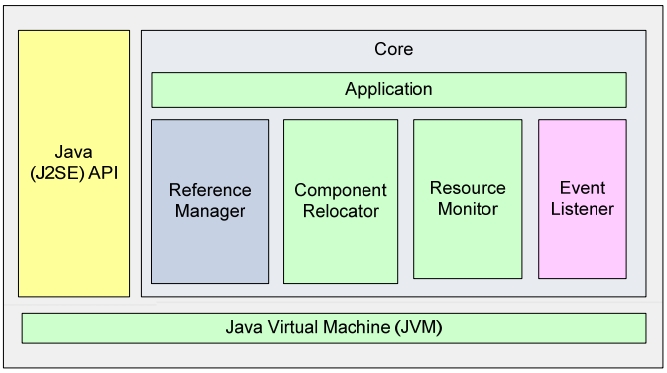

Figure 3.3. Module layout for the disconnected operations framework

With reference to the framework, the objects are notified about disconnection or reconnection via event-notification mechanism. The resource monitor determines the status of the network bandwidth, which is used to determine whether the application should switch from connected mode to disconnected mode. However this information is utilized when the framework is actively monitoring the resources (particularly for involuntary disconnection) and predicts the status, to allow the application to prepare for the adverse circumstances. For the time being we are focusing our efforts on the remedies for disconnections. We plan to consider prediction of involuntary disconnections as our future research work.

3.2.1.1.5 Prototype Implementation

We are implementing a prototype application for patient monitoring and diagnosis service (which executes in a Grid environment) along with the framework libraries in order to verify the feasibility of our proposal. An illustration diagram is shown in figure 3.4. This implementation is based on [26]. The module layout of the framework along with the application is shown in figure 3.3. The *component relocator* deals with downloading the implementation (.class) files at the client, from the server. Where as the *reference manager* maintains the instantiated references after disconnection so that the system may return to its original state upon reconnection. It is also

responsible for swapping the references (remote and local) in order to maintain service availability. The *event manager* (a wrapper of the support provided by JVM) deals with triggering events in the framework. Where as the *resource monitor* determines the state of the network. It is this module which will facilitate the process of predicting a disconnection. It may be noted that the framework comprises of two subsystems; one operating at the server end and the other at the client end.

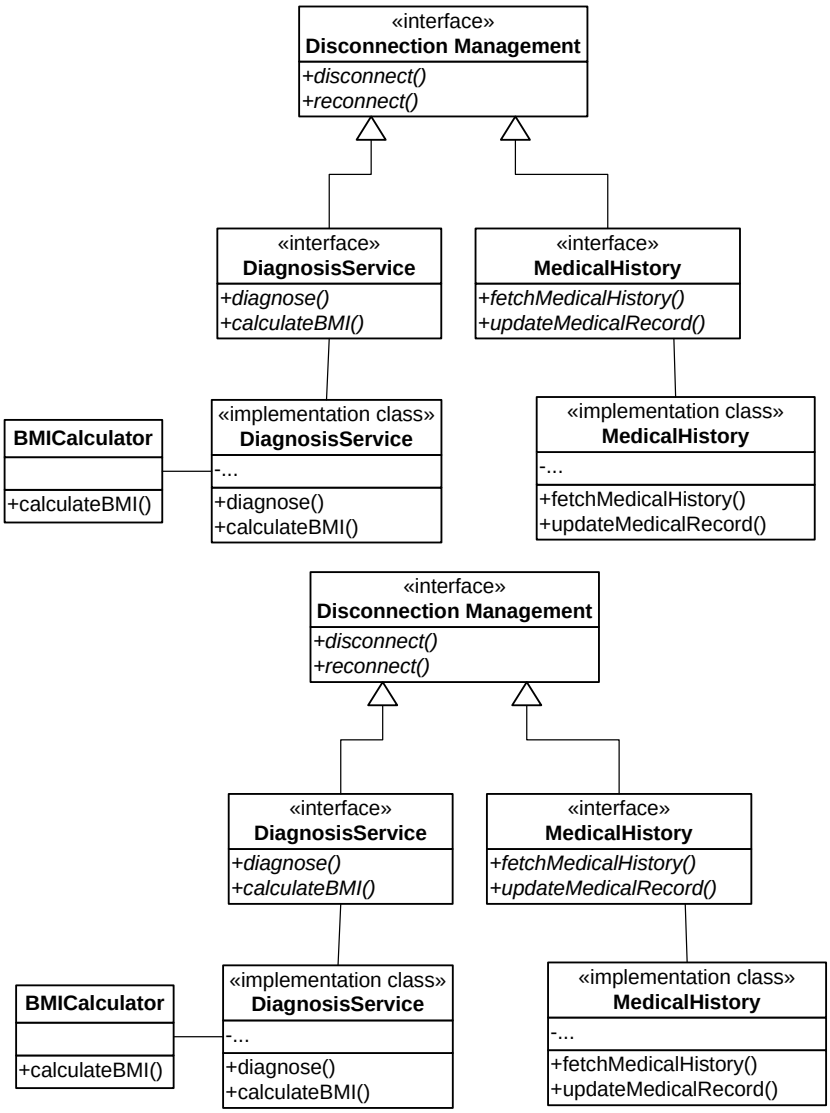

Figure 3.4. Prototype Implementation

Unlike [27], our approach being simple and discreet avoids the computational overhead required to determine the component distribution in different circumstances.

This is primarily due to the application aware approach, which allows the developer to determine the application policies.

3.2.1.1.6 Related Work

A substantial debt is owed to Coda [28], which extends AFS [29]. The authors were among the first to demonstrate that client resources could be effectively used to insulate users and applications from the hurdles of mobile information access. Coda treats disconnection as a special case of network partitioning. The client may continue to use the data present in its cache, even if its disconnected. However it is the user profile that determines which files are to be maintained in the cache. In disconnected mode, the client accesses files from the cache only. The cache-misses are considered as failures (such as file not found error). However when the client reconnects, the cache is updated.

Odyssey [30], inspired by Coda [28], proposed the concept of application-aware adaptation. The essence of this approach is to have a collaborative partnership between the application and the system, with a clear separation of concerns. Functionality that is implemented monolithically must be split between the operating system and the individual applications. Odyssey encourages the operating system to sense external events and to monitor and allocate resources. Whereas the role of the application, is to adapt to the changing conditions, by using the information and resources provided by the operating system. We extend this notion by suggesting that the middleware/framework must provide the means for adaptation as well. Whereas the application is left with the sole task of sharing the application policy; which components of the application are to participate in the process of adaptation (and any application specific processing that might be required before and after the adaptation).

Ficus [31] is a distributed file system that deals with disconnected operations. They utilize an optimistic replication scheme which permits the update of local copies and determines conflicts at reconnection. The primary focus of this project is on high degree of data availability in the face of an involuntary disconnection. FarGo-DA [25], an extension of FarGo, a mobile component framework for distributed applications proposes a programming model with support for designing the behaviour of applications under frequent disconnection conditions. The programming model enables designers to augment their applications with disconnection-aware semantics that are tightly coupled with the architecture, and are automatically carried out upon disconnection.

Rover [32] is a software toolkit for developing mobile aware applications. The toolkit presents two primary abstractions; the relocatable dynamic objects (RDOs) and queued remote procedure call (QRPC). The RDO allows dynamic loading at the client from the server. The QRPC presents a communication scheme that allows applications to make non-blocking remote procedure calls, even in the face of a disconnection. Such requests are queued and transferred upon reconnection. Rover utilizes a

caching mechanism to deal with disconnections. This might be expensive when considering resource constrained devices.

Other efforts such as [33] have also proposed middleware frameworks for dealing with disconnected operations. Such efforts suggest that techniques of caching, hoarding, queuing remote interactions, replicating may prove beneficial in resolving the problems of disconnected operations. However these endeavours either sacrifice their accuracy or the service delivery time.

In another research endeavour [27] the authors suggest that the critical difficulty in achieving the task (of distribution of software components) lies in the fact that determining a software system's deployment that will maximize its availability is an exponentially complex problem. They propose an approximation algorithm for this problem. Their study is guided by the observation that, in these environments, a key determinant of the system's ability to effectively deal with network disconnections is finding the appropriate deployment architecture. While the redeployment problem has been identified in their literature, its inherent complexity has either been ignored, thus making it infeasible for any realistic system, or highly restricted, thus reducing the solution's usefulness. We resolve this problem by suggesting an approach of application aware adaptation, which allows the designer to determine the application policies and thus avoid the constraints all together.

Chapter 4: Knowledge Management for Autonomic Middleware

The key concept to introduce autonomic knowledge management in the MAGI middleware architecture is an autonomic component. The knowledge management component is itself composed of autonomic components or elements [56] which are responsible for managing their own behavior in keeping with the relevant policies, and also for cooperating with other autonomic components and elements to achieve their objectives.

4.1 Structure of Autonomic Components in MAGI

As stated in the previous section, the autonomic component in the MAGI middleware are responsible for both local self-management i.e., managing their own behavior in keeping with the relevant policies and cross-component collaboration i.e., cooperating with other autonomic components to achieve their objectives. The basic structural elements of a typical autonomic element in MAGI middleware are depicted in the Figure 4.1.

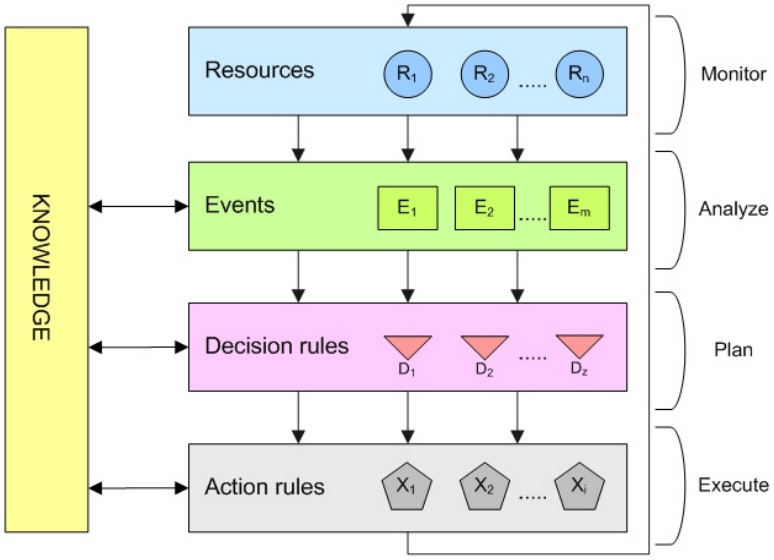

Fig. 1. Structure of an Autonomic Component in MAGI

The autonomic component model consists of following layers:-

- **Resources:** A resource can be anything, from a physical or hardware asset like a CPU, memory etc. to a virtual or software or logical asset like an application or service. The resource model is of central importance in representing what is being managed by the autonomic component.
- **Events:** The event signifies a change in the state of a resource, policy or component, which can be either minor or major depending upon the nature and goals of a particular component.
- **Decisional Mechanisms:** Decision rules are used for deducing a problem based on the information received from various events. The problems can be deduced using a single event or inferring from a group of events, based on a single occurrence or based on a history of occurrences. The component then makes plans to rectify this problem or optimize some functional/behavioral aspects, depending upon the policies and internal and external knowledge of the component.
- **Action Mechanisms:** Action rules are used to carry out execution mechanisms to bring about these changes in line with the desired goal of the component, e.g., running dedicated algorithms for problem solving etc.

This manner of structure facilitates in building up a control loop by employing the monitor, analyze, plan and execute cycle, which is of key significance for autonomic

behavior [57]. This facilitate in the compartmentalization of the management procedure, by breaking the functionalities into gathering the knowledge, analyzing it, making a plan of action and the executing that plan.

The type of model requires the presence of a common knowledge base that contains the knowledge about a problem space that is shared among the four elements of the model. This shared knowledge includes such features as network information, system logs, performance metrics, and policies that are relevant to the problems.

4.2 Architecture of the Knowledge Management component

The knowledge management component's job is to manage & analyze **i**nformation regarding all its constituent components, make plans based on that information and then execute them. It accomplishes this takes by making use of intelligent decision-making, taking inferences from client's and the mobile device's profile, the type and nature of the submitted job and the type and nature of the Grid service etc. and also takes into account the policies that rule the undertaking of the whole operation. It then uses the results for the self-management of various factors of the operation of the middleware, like QoS, fault-tolerance and work-load etc.

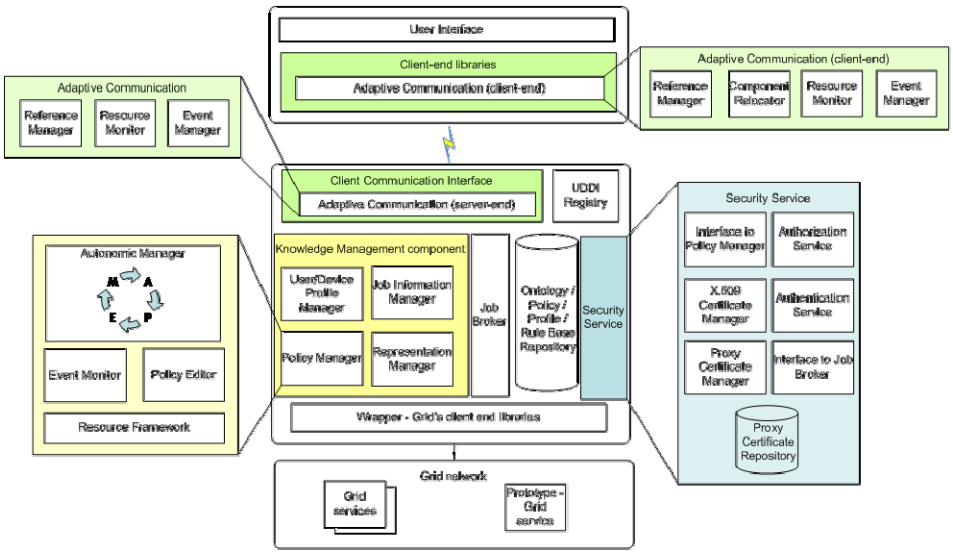

Fig. 2. Detailed architecture of the MAGI middleware

Figure 16 shows the detailed architecture of the complete MAGI middleware, but the scope of this dissertation is only focused on the Knowledge Management compo-

nent. The description of various constituent components facilitating the working of knowledge management is as follows:-

4.2.1 User/Device Profile Manager

As evident from the name, it manages the profiles of both the users or clients of the middleware service and also the profiles of the mobile devices that they use to access the Grid services through the middleware.

It gets the user/device profile from the client if it's already not present in the system repository. This transfer of profile is handled in cooperation with the security mechanisms in place in the middleware, to manage the issues of privacy and authentication.

The client may or may not send the contextual information regarding the job that it wants to submit to the Grid embedded in the profile. If such information is present in the profile, the policy manager will execute the job keeping in view the requirements described in the profile regarding that job. If there is some conflict as a result of the client's requirements, e.g., authorization issues, then the Autonomic Manger will try to resolve that conflict taking guidance from relevant policies present in the repository for this purpose. In case of the absence of any requirement for the operation of the execution of a particular job, it will follow the default policy associated with the particular category of the job or request.

4.2.2 Job Information Manager

The job of this manager is to handle the different states of a job submitted by clients. Again, the management of different states of a job depends upon their presence in a well-define manner in the profiles sent by the clients. The MAGI middleware cannot assign the job states on its own. So clients have to send the different states of the job in their profile, e.g., the client might send a job whose results they want back in certain intermediate states of completion. So the middleware can send its requests to the Grid in intermediate steps as requested in the job's states if the nature of the job and Grid service permits that, otherwise it may submit the complete job to the Grid service and just send back the received results to the clients in accordance to the specifications of the states.

The job information manager also monitors the performance metrics associated with a job e.g., time taken to provide a service or result back to the client i.e., performance analysis, scheduling, collections of logs or system statistics and fault detection and fault handling mechanisms in case of some problems occurring with the jobs that have been submitted to the middleware. Also, the clients can be given an estimated time of completion and resource estimation etc. based on predictions from historical data, logs and statistics collected by the manager.

4.2.3 Representation Manager

The job of this manager is to handle the formatting of job results according to device profile and sending it to the device. Different mobile and wireless devices have different display options. Devices like notebook computers have almost no limitation on the types of visual representational schemes that can be utilized for them, but for devices like PDA's and cell phones, the display options are heterogeneous and usually constrained. For example, if mobile device is a cell phone, we can automatically generate a WML (Wireless Markup Language) page with the results and their brief description and send it to the device.

For this purpose, UIML is utilized (User Interface Markup Language) [58], a markup language extension of XML that promotes the creation of Web pages that can be viewed on any kind of interface device, from PC monitors to smart phones to PDAs. Using UIML style sheets, Web content can be created once without knowing specifically which devices it will be viewed on. A developer uses UIML to describe elements of the user interface -- such as menus, buttons and input boxes. A programmer then can write applications that rely on UIML to get the content to different devices.

The skeleton of a UIML document is shown in Figure 17.

```
  <?xml version="1.0">
<!DOCTYPE uiml ... "uiml2_0g.dtd">
<uiml>
    <interface>
              <structure>   ...</structure>
              <style>       ...</style>
              <content>     ...</content>
              <behavior>    ...</behavior>
      </interface>
      <peers>
              <logic>       ...</logic>
              <presentation>...</presentation>
      </peers>
  </uiml>
```

Fig. 3. Skeleton of a UIML document

By making use of this technology, we can rid ourselves of the need to have multiple source code families in order to deploy interfaces on multiple devices, and relieve ourselves from the burden of having to manage interface content depending on what devices the content will be viewed on, and eliminates the risk of developing device-specific interfaces for a device that may not be on the market in the future.

4.2.4 Policy Manager

The Policy Manager is the core component that incorporates the autonomic behavior in the middleware by facilitating the working of Knowledge Management processes and tasks.

The foundation of its autonomic management is based upon the Autonomic Manager sub-component. It makes policy decisions based on a set of policies created by the domain experts to influence a particular component or run-time entity. When a managed entity requests some assistance, the Autonomic Manager evaluates all relevant policies and returns a decision. These decisions are usually in the form of a result (value) but can also be a configuration profile (property). This action can also be initiated from the Autonomic Manager without a request from the managed resource. These decisions can be in the form of an action (a sequence of executions) or a configuration profile (setting properties). It also deals with the issues of job states, client have to send a policy with along with the job requests that will instruct the middleware how to deal with job related issues like defining states, whether to initiate asking for intermediate results from the Grid or job status from the Grid, etc.

The common format used for writing policies is XML. Policies that conform to this language include the following key attributes: scope (the domain of the policy), condition (policy trigger) and decision (result, action, configuration profile). A policy describes the guidance that influences the behavior of a managed module or entity. Its scope of influence is often a particular instance of a job. The Policy Editor sub-component deals with the dynamic insertion, search, modification, access, storage, versioning and removal of the policies in the Knowledge Management component.

The Event Monitor sub-component has a monitoring and even notification role. It can also initiate communication with the AM in case of a fault or problem i.e., it utilizes both periodic and polling-based approaches towards even notification.

The Resource Framework is a model for identification of all the resources that the Event Monitor will be observing.

The Autonomic Manager sub-component also incorporates predictive autonomicity for fault-tolerance and self-healing in the middleware, using historical contextual knowledge. In the later sections various prediction approaches will be discussed like Step-Wise Regression, Rule Induction and Artificial Neural Networks and it will be discussed which is the most suitable for the optimum performance and look-ahead trade-offs. Maintaining the historical contextual knowledge is useful also because if a fault or problem happens repeatedly, or if a certain classification of jobs is submitted frequently by a client, instead of doing all the calculations again, the Autonomic Manager just takes the results from the system repository.

4.2.5 System Repository

The System Repository is basically a Policy/Profile/Ontology/Context database. It is a system-wide repository for handling policies, user and device profiles, historical and contextual information of the working of the system as well as configuration profiles for various jobs, components etc. It maintains policies defined by clients, middleware developer or Grid services. If origin of a policy is a client or Grid service, then it is subject to the middleware authorization regulations (part of the security infrastructure), in other words, MAGI's policy supersedes the client policy.

4.3 Implementation Overview

The main purpose of using the Policy Manager and its Autonomic Manager is to provide the guidance to the functional behavior of components by extracting and externalizing functional logic into sets of rules. Modification in the policies does not affect the associated data and applications, resulting in a higher level of maintainability, variability and manageability. Most current policy systems are domain and environment specific, and therefore require domain and environment specific information to execute the policies. The Policy Manger of the Knowledge Management component provides a flexible and usable system by utilizing component technologies and separates the process of executing a policy into various sub-processes or components. Each sub-process or component can be developed independently, configured independently, and reused**.** Some of these are discussed below.

4.3.1 Conflict Handling

In any system hosting more than one policy for achieving a set of goals, some ambiguousness or conflict of interest may arise. The logic used for conflict resolution is a set of instructions which can resolve conflict among various outcomes of executing a policy by determining the condition and action attributes of the policies, verifying that the actions specified do not conflict and checking to find redundant policy rules within the system.

4.3.2 Priority Handling

The logic used for priority handling is a set of instructions which determines the priority of each individual policy rule in the policy, and the priority ranking of different policies if the usage of more than one policy is required in the outcome of a job or request.

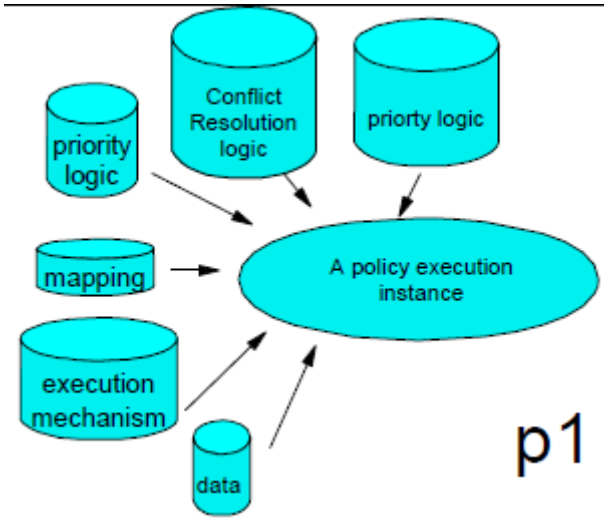

Fig. 4. The Autonomic Manger is influenced by various Policy-related processes which are dynamically re-configurable

4.3.3 Policy Mappings

Policy mapping is a very essential aspect of this operation, as it is used to connect the conditions and actions specified in a policy to the actual data or functions which will provide the new data or execute the relevant actions.

```
if (observedValue("pmdo1", "throughput",
"minutes", 10) < 95% of
expectedValue("pmdo1", "throughput"))
then (createAlert("logEntry", "pmdo1
throughput is below 95% of specified value…"))
```

Fig. 5. Example of execution logic created from a policy

Policies in the MAGI middleware do not embody executable code, so the mapping process involves converting the semantics of the policy into execution constructs like execution logic and database entries and other resources etc. A policy for a different job classification may require different mapping; while its logic expressed as policy may be the same, the data, class objects or device resource model may be different.

4.3.4 Knowledge based Execution

The knowledge required for the execution of the actions required for achieving the difference objectives defined in the policies is stored in the system repository, as

discussed earlier. Depending upon that, the Autonomic Manger uses the sets of instructions, rules and execution algorithms available there to control the way the policy is carried out.

A typical execution begins by taking a policy document associated with the particular job or request, from the system repository if it is already present there; otherwise it is supplied by the client along with the job or request. A set of resource mapping definitions expressed in XML are then used to map the policy. The output of the mapping is an executable rule-set or operational logic which is then executed by the Autonomic Manager to carry out the task.

4.4 Related Work

Not much work has been done in the area of integration of mobile computing and Grid computing domain in the past years, and among the notable projects, much of them are only concerned with one or more aspects in an ad hoc manner. Following is a summary of the most prominent projects in this research field.

4.4.1 GridBlocks

GridBlocks [59] builds a Grid application framework with standardized interfaces facilitating the creation of end user services. A portal based gateway, GBPortal, provides a web interface for Grid resources, using the Java Servlet and Java Server Pages (JSP) technology. Because of this approach, this solution provides a web-interface for interaction with the Grid services just like accessing a dynamic website. The user can submit jobs to grid without any local software installations. As a consequence of this approach, only one-way communication is possible, i.e. from the client-end to the server-end, where it the client who initiates the request.

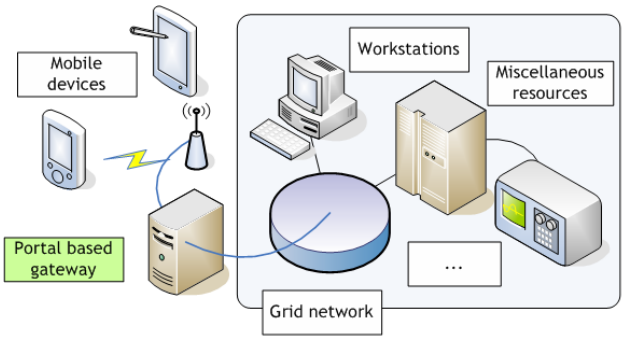

Figure 4.6. Portal-based Architecture of GridBlocks

For security, they are inclined towards the MIDP specification version 2 which includes security features on Transport layer. They advocate the use of propriety protocol communication protocol and state that performance of SOAP (Simple Object

Access Protocol) on mobile devices maybe 2-3 times slower as compared to a proprietary protocol. But in my view, proprietary interfaces limit interoperability and extensibility, especially to such a heterogeneous domain like personal mobile devices. Furthermore, open standards is a salient feature of the autonomic computing paradigm.

For knowledge management, they use StorageBox [60], an open source Personal Database System that features an information management system running on top of MySQL RDBMS.

4.4.2 Mobile Agent based Platform

This project [61] uses the mobile agent paradigm to take care of all the details to allow mobile users to access distributed resources in a effective way. It focuses on providing this access transparently and keeping the mobile host connected to the service. Though they have to improve upon the system's security, fault tolerance and QoS, their architecture is sufficiently scalable.

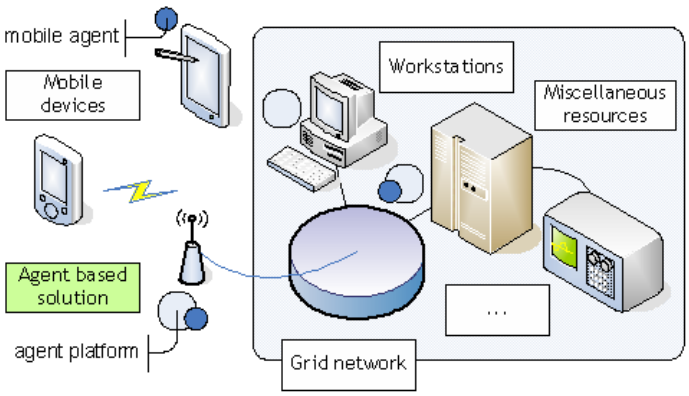

Figure 4.7. Mobile Agent based approach

However, this approach results in the requirement of a mobile agent platform in the Grid network, for the mobile agents to execute on. This approach has not been supported by the Grid community and hence this addition is not compliant with the Grid standard. Also, the authors acknowledge the fact that communication overhead is incurred as the mobile agent will have to move (back and forth) between the client and the Grid for intermediate interactions with the client-side.

For the purpose of knowledge management, the authors don't follow any particular methodology as each agent carries the relevant information regarding its operation with it. This procedure can be very cumbersome for applications and requests that demand a large amount of data transfer.

4.4.4 Signal

Signal [62] proposes a mobile proxy-based architecture that can execute jobs submitted to mobile devices, so in-effect making a grid of mobile devices. A proxy interacts with the Globus Toolkit's MDS (Monitoring and Discovery System) to communicate resource availability in the nodes it represents. The proxy server and mobile device communicate via SOAP and authenticate each other via the Generic Security Service (GSS) API. The proxy server analyzes code and checks for resource allocation through the MDS. After the proxy server determines resource availability, the adaptation middleware layer component in the server sends the job request to remote locations. Because of this distributed execution, the mobile device consumes little power and uses bandwidth effectively. Also their efforts are more inclined towards QoS issues such as management of allocated resources, support for QoS guarantees at application, middleware and network layer and support of resource and service discoveries based on QoS properties.

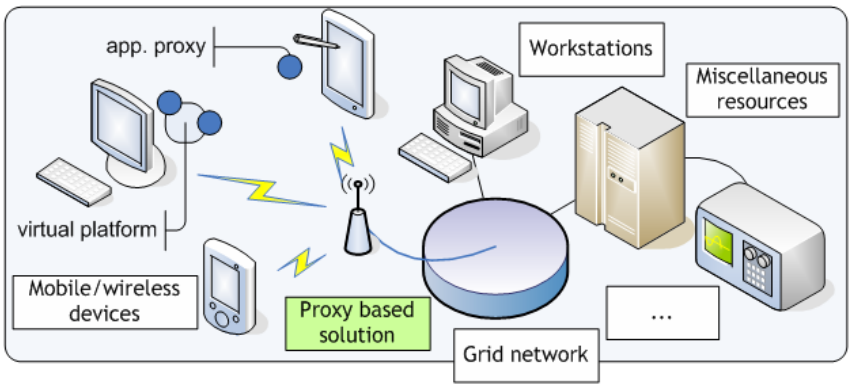

Figure 4.8. A Proxy-based approach

However, the authors here propose that the client platforms should be part of the Grid. Consequently the applications will be using an API compliant with the Grid standard. If the Grid community doesn't provide a light-weight interface for the applications (to be run on mobile devices) we will be back to square one; communication overhead or even impossibility for the client devices as a result of adhering to the standard. Also, the authors claim that the client application doesn't interact with the Grid service directly; a proxy/gateway is elected among the devices to act as a virtual platform for the rest of the application proxies. This again arises contentions for the selection criteria and the effect of load on the effected device and the rate of transfer of this responsibility.

For knowledge management, they use Service Data Elements (SDEs) [63] of OGSA (Open Grid Services Architecture) to define information parameters. This is a mechanism of publicly expressing the available state information of a service through a known schema. This concept is not limited to Grid services. Any stateful Web service can declare its publicly available state information through service data concepts.

Furthermore, the SDE declaration is similar to the xsd:element declaration, but the SDE declaration is a restriction to the xml schema xsd:element that uses only six properties of the xsd:element declaration (annotation, name, type, occurs(minOccurs, maxOccurs), nillable, and the open attribute model) and adds two new attributes "mutability" and "modifiable" using the open attribute model.

All of these research efforts don’t envision the incorporation of self-management and the capabilities of autonomic computing paradigm in their systems.

Chapter 5: Security

5.1 Introduction

The Mobile-to-Grid Middleware Environment can be visualized as in Figure 2. Mobile device users use different mobile devices to interact with the MAGI using security protocols as defined by the MAGI middleware. MAGI in turn communicates with the grid using the standard GSI protocols. In what follows, we will use $A_1, A_2, \ldots$ for the mobile clients, $M_1, M_2, \ldots$ as the MAGI instances and $G_1, G_2, \ldots$ as any generic grid service, with the subscripts omitted if a general instance of that category is under discussion.

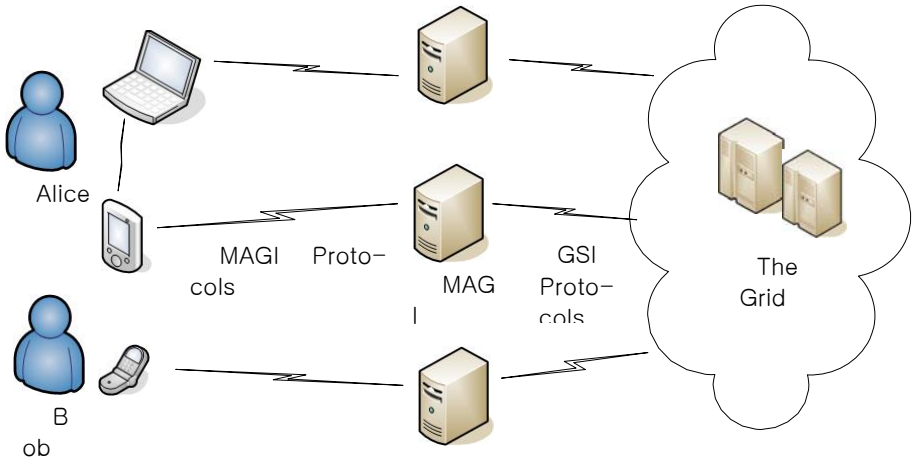

Figure 1. The Mobile-to-Grid Middleware Environment

The mobile client (A) and MAGI (M) contain Client Security Manager and MAGI Security Manager respectively. As the name indicates, these two managers will be responsible for secure communication and security decisions. We will abuse the notation for A and M and use it interchangeably for both the mobile device and MAGI, and the security managers. Figure 3 and 4, show the modular diagrams of the security managers A and M .

Both the managers have some core services and some additional optional packages for added security. The main service is the Authentication and the Privacy Service which provides authentication through digital certificates and data integrity and confidentiality through encryption. This service is present both in A and M . The Authorization and the Delegation services are not part of the Client Security Manager as they are intended for authorizing the client and job delegation purposes respectively. The trust manager helps A and M to calculate the trust they can put on each other based on their

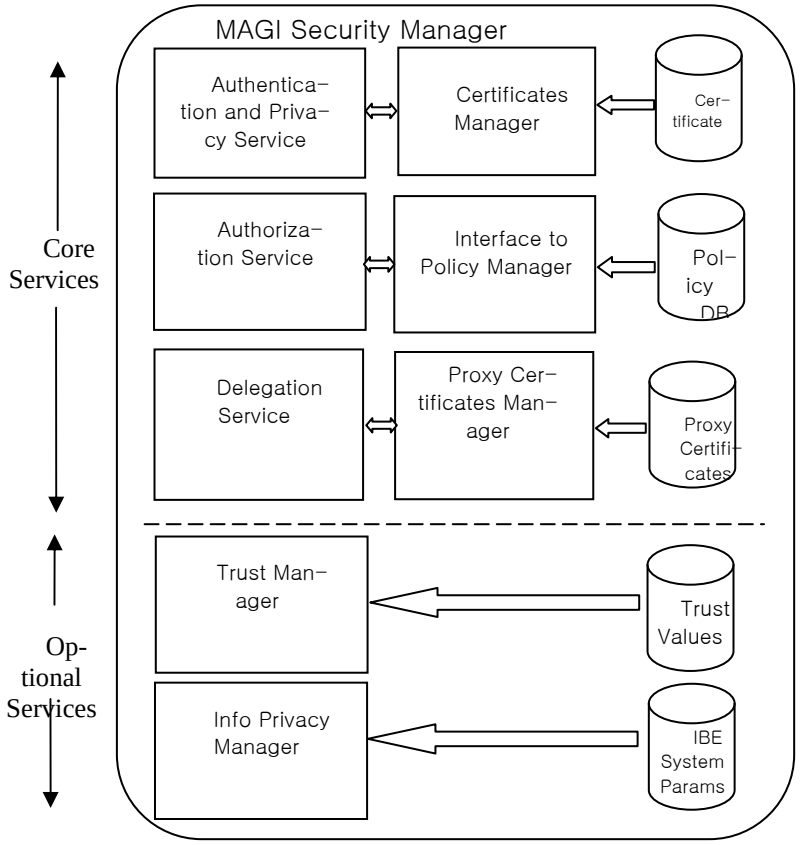

Figure 2. MAGI Security Manager

past experiences as well as the opinion of the other peers. This module is useful in a very large scale system and hence is only optional for normal sized networks. The Information Privacy Manager helps to keep the results obtained from the grid services in encrypted format on MAGI and enables MAGI to search on this encrypted data with some feed back from the mobile client without decrypting the data. This module like the previous one is also optional and is meant for very high security requirements.

We will introduce and explain each one of these modules in the next sections by giving a detailed account of the underlying protocols, algorithms and mechanisms employed.

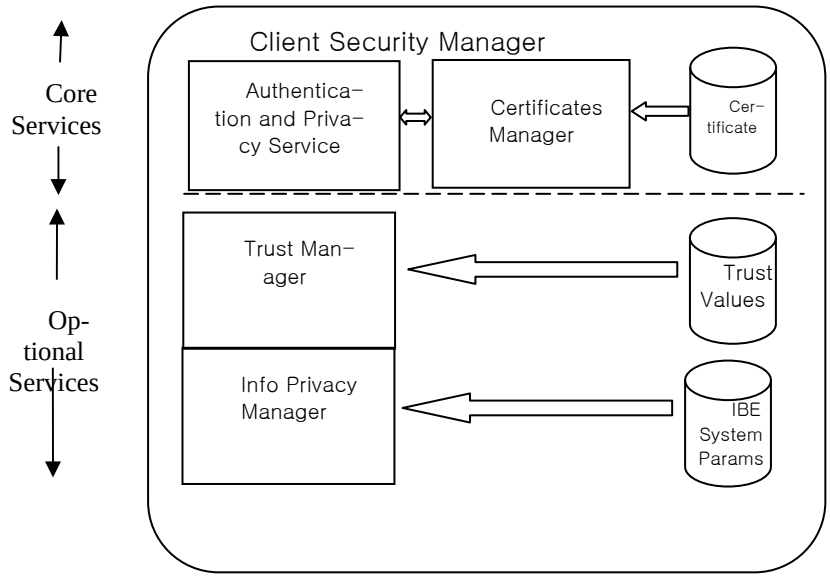

Figure 3. Client Security Manager

5.2 Authentication and Privacy Service

The responsibility of the Authentication and Privacy Service can be divided into two main objectives:

1. To ensure secure communication (authenticated and perhaps confidential) between A and M .
2. Safeguarding the data residing on the mobile device under potential threat of the device been stolen.

The Grid Security Infrastructure is based on public key cryptography mainly deployed using the RSA public key cryptosystem [8]. However key sizes in the RSA scheme are large and thus computationally heavy on handheld devices such as PDA's, mobile phone's, smart phones etc [9]. The Elliptic Curve Cryptography (ECC) [13] based public key scheme can be used in conjunction with a symmetric key system such as, Rijndael's Advanced Encryption Standard(AES)[14], for mobile access to Grid which provide the same level of security as RSA and yet the key sizes are a lot smaller [9]. In what follows A_{priv} and A_{pub} denote the private and public key of

Alice A. The private and public keys of M and G are defined likewise. The Encryption Algorithm and Decryption Algorithm will be denoted by $E(.)$ and $D(.)$ respectively, with the key specified as a subscript.

5.2.1 Authentication

We assume that there is a CA that issues Elliptic Curves based certificates, known as the Elliptic Curve Digital Signature Algorithm (ECDSA) [15] certificates. Both A and M have these certificates and they use these certificates to authenticate with each other. M on the other hand, also has a RSA based X.509 certificate that can be used to communicate with the Grid Services. This may give rise to compatibility issues during job delegation. We deal with this problem in the section 2.3. The protocol to authenticate A to M is described as follows:

Protocol 1: *Authenticate*(A, M)

INPUT: A and M 's ECC certificates
OUTPUT: accept or reject

1. A sends a request for connection to M.
2. M acknowledges and asks A for its ECDSA certificate.
3. A sends the certificate to M and M checks its validity.
4. M sends a challenge cyphertext to A.
5. A decrypts the message using its private key and sends the Response to M.
6. M decrypts using A's public key.
7. If the response is right than accept otherwise reject.

Table 1. The Authentication Protocol

We can easily enable mutual authentication by running the above protocol with the roles interchanged. So the mutual authentication protocol is:

Protocol 2: *MutualAuthenticate*(A, M)

INPUT: A and M 's ECC certificates
OUTPUT: accept or reject

1. *Authenticate*(A, M)

2. If $Authenticate(A, M)$ =reject than halt, else continue
3. $Authenticate(M, A)$
4. If $Authenticate(M, A)$ =reject than return reject otherwise
   accept

Table 2. Mutual Authentication Protocol.

5.2.2 Data Confidentiality and Integrity

Data integrity can be provided by using encryption. After authentication M generates a secret key of Rijndael's Advanced Encryption Standard algorithm (AES) [14] and sends it to A by encrypting it with the latter's public key. A and M can then use this key to encrypt and/or decrypt any further messages between the two parties. This key can be destroyed when a session is over.

The protocol is as follows:

Protocol 3: $GenerateSessionKey(A, M)$

INPUT: A's Public Key
OUTPUT: AES Session Key K_{sess}

1. M generates an AES secret key K_{sess}.
2. M computes $E_{A_{pub}}(K_{sess})$ and sends it to A.
3. A computes $D_{A_{priv}}\left(E_{A_{pub}}(K_{sess})\right) = K_{sess}$.

Table 3. Session Key Generation Protocol.

Our choice of using Rijndael's AES is due to its ability to provide faster computation with low memory requirements [14]. Alternatively we can use the Elliptic Curve Diffie Hellman Key Exchange protocol to exchange the AES key. Data integrity can be provided by using standard Message Digest Algorithms like SHA-1 [16].

5.3 Key and Data Safeguarding

A's private key is stored in a file in the mobile's storage which is encrypted via a password. To use this key, the user must enter the pass phrase required to decrypt the file containing their private key. If a device is stolen, the data residing in the mobile device can be compromised. Even if the key is protected by the password, the adver-

sary has access to the disk containing all the data. The adversary would still have access to the data and the file containing the key. To protect this situation, we require A to encrypt the data on it using a newly generated AES key and then apply the notion of secret sharing schemes to produce two or three shares of this key. One is sent to M, the second is kept by A in the same device and the third on another of Alice's device. Now the adversary cannot decrypt the data since it does not have the other share of the key without which it cannot learn anything about the key. It would also not risk to communicate with M, which would certainly yield his location. In the next section we give a detailed account of secret sharing schemes and propose a construction that is suitable for the low-end mobile devices. Readers not interested in the detailed mathematical construction can jump to section 2.2.7 after reading the next section.

5.3.1 Background on Secret Sharing Schemes

Shamir[17] and Blakeley[18] independently proposed the notion of a secret sharing scheme. A (k, n) secret sharing scheme (threshold scheme) is a method of dividing a secret s into n shares $s_1, s_2, \ldots, s_n$ and giving a unique share to a set of n participants $\mathrm{P} = \{P_1, P_2, \ldots, P_n\}$ such that the knowledge of any k or more shares makes the secret s easily computable but knowledge of $k-1$ or less shares does not reveal any information about the secret s. Here, k is called the threshold of the scheme. The secret s is chosen by a Dealer $D \notin \mathrm{P}$ and gives the share s_i to the participant P_i over a secure channel.

An elegant feature of Shamir's Scheme is that it is unconditionally secure. A consequence of this property is that the size of shares per participant be at least the size of the original secret [19]. This poses a fundamental limit on unconditionally secure secret sharing schemes. Against resource bounded adversaries this lower limit can be relaxed by introducing computational security. Krawczyk [20] was the first to discuss computationally secure secret sharing schemes by reducing the size of shares to approximately $|s|/k$. However our discussion will be limited to unconditionally secure schemes. Although Shamir's secret sharing scheme is perfectly secure, however it is not free of cheating. Tompa and Woll[21] presented a possible scenario in which a participant or a group of participants can send a false share and hence get the correct secret whereas the honest participants will be deceived with the wrong secret. This situation will go undetected in Shamir's scheme described above. Tompa and Woll[21] first showed how to prevent this form of cheating keeping in mind the unconditional security assumption. An outcome of their scheme is that the size of shares should grow with the probability of cheating detection. This was shown to be undeniable in [22] where the authors gave lower bound on the size of shares. This bound was improved by Kurosawa et al in [23]. Another undesirable property of cheating

detection schemes is that even though cheating is detected (or in some other schemes cheaters are identified) the cheaters still recover the secret whereas the honest participants don't get the correct secret. Tompa and Woll showed how to prevent this situation. They encoded the secret as a sequence, where only one element of the sequence is assigned the actual secret and the other elements a dummy value. These elements are then divided into shares and given to each participant. There are a couple of disadvantages in their method. Firstly, each participant has t shares to keep (equal to the length of the sequence). Secondly the probability of the cheaters to get the secret is $\approx 1/t$ which can be made less by making the number of shares large, where each share is at least the size of the original secret. The goal of this paper is to increase this probability without considerably increasing the size of shares. For a detailed account of the related work, see [24].

We propose a secret sharing scheme to prevent cheating with high probability using individual shares whose total size is roughly the same as the size of the secret. The basic theme is to divide the shares into parts, introduce random numbers and shuffle them with the shared pieces in such a way that the participants don't know the exact construction of the shares unless at least k of them pool their shares together. The participants pool the individual parts one by one and the notion of hash functions as in [25] is used to detect cheating at every stage. The resulting scheme prevents cheating against any set of $k-1$ cheaters. Introduction of a one way hash function of course takes away the unconditionally secure property of the Shamir Secret Sharing Scheme, however it can help in reducing the size of the shares per participants. The scheme can successfully detect cheating provided there exists a collision free one way hash function. The main contribution of our work is not to detect the cheaters but to prevent them from attaining the secret under our communication model.

5.3.2 Proposed Scheme

Before going on to the proposed scheme, we revise the notion of permutations which will be needed to describe our scheme. For a positive integer t, let $[t] = \{1, 2, \ldots, t\}$. A *permutation* σ of $[t]$ is a one-to-one correspondence from $[t]$ to $[t]$. Let R_t denote the set of all permutations of $[t]$. The identity of R_t denoted by ι is a permutation defined by the rule $\iota(x) = x, \forall x \in [t]$. The composition of two permutations σ and τ is defined as $\sigma\tau(x) = \sigma(\tau(x)), \forall x \in [t]$. The inverse $\sigma^{-1} \in R_t$ of the permutation $\sigma \in R_t$ is defined by $\sigma\sigma^{-1} = \sigma^{-1}\sigma = \iota$. A *ranking function* assigns a unique integer in the range $[1, t!]$ to each of the $t!$ permutations in

R_t. The corresponding *unranking function* is the inverse, which given an integer in the range $[1, t!]$ returns the permutation with this rank.

Next we define a few functions which will be required in our scheme. Let p denote a positive integer and let $x \in Z_p^+$. Let $|p|$ denote the number of bits in the binary representation of p.

Definition 1 The concatenation of the positive integers x and y is a number obtained by appending the binary representation of y after the binary representation of x. We denote it by $con(x, y)$.

This definition can be extended to more than two arguments in a natural way.

Definition 2 For a positive integer t, the function $split(x, t, p)$ is an ordered t-tuple defined by,

$$split(x, t, p) = (x_1, x_2, \ldots, x_t)$$

Such that,

$$|x_i| = \lceil |p|/t \rceil \text{ for } 1 \le i \le t-1,$$
$$|x_t| = |p| - \lceil |p|/t \rceil (t-1),$$
$$\text{and } con(x_1, x_2, \ldots, x_t) = x.$$

Definition 3 Given an ordered t-tuple of elements $x = (x_1, x_2, \ldots, x_t)$, we say that the ordered t-tuple $O_x(\sigma) = (x_{\sigma(i_1)}, x_{\sigma(i_2)}, \ldots, x_{\sigma(i_t)})$ is a reordering of x according to the permutation σ, if

$$1 = \sigma(i_1) < \sigma(i_2) < \ldots < \sigma(i_t) = t$$

where $i_1, i_2, \ldots, i_t \in [t]$ distinct from each other and *not necessarily* in ascending order.

We denote the initial ordered t-tuple $x = (x_1, x_2, \ldots, x_t)$ by $O_x(\iota)$.

Now we are set to describe our scheme. Continuing with our previous notation, let P be the set of n participants $\{P_1, P_2, \ldots, P_n\}$, let $S = \{0, 1, 2, \ldots, s-1\}$ be the set of possible values of the secret and let k be the threshold of the scheme. The scheme will be shown in two steps: Secret Generation Phase and Secret Recovery Phase. We

assume an honest Dealer D and a secure channel between the Dealer and each participant in the Secret Generation Phase.

5.3.2.1 Secret Generation Phase

This phase is carried out by the Dealer D. It is assumed that for a given integer t, the set of all permutations R_t is ranked according to a ranking function.

1. Choose a prime $p \geq \max(s, n, t!)$.
2. Select the secret S from the set of non negative integers $\{0,1,2,\ldots,s-1\}$.
3. Select random integers $a_1, a_2, \ldots, a_n$ from the integers $(0, p)$ to construct the $k-1$ degree polynomial:
$$f_1(x) = S + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$$
4. Compute $s_i = f(i)$ for $1 \leq i \leq n$.
5. Select a positive integer t, and compute $b = \lceil |p|/t \rceil$.
6. For $1 \leq i \leq n$,

6.1 Compute $split(s_i, t, p)$. If $\lceil |p|/t \rceil \neq |p|/t$, then select a random positive integer q such that $|q| = \lceil |p|/t \rceil t - |p|$ and compute $s_{it} = con(s_{it}, q)$. Now concatenate the first $t-1$- pieces as $S_i = con(s_{i1}, s_{i2}, \ldots, s_{i(t-1)})$.

6.2Let $e_{i1} = s_{it}$ and select $t-1$ random positive integers $e_{i2}, e_{i3}, \ldots, e_{it}$ such that each one is of size $\lceil |p|/t \rceil$. Create the t -tuple $O_{s_i}(\iota) = (e_{i1}, e_{i2}, \ldots, e_{it})$.

7. Choose a one way function $h(.)$ with the condition that $h(.) < p$ and a positive constant c such that $1 \leq c < p$.

8. For $1 \leq j \leq t$, compute $T_j = \sum_{i=1}^{n} h(e_{ij}) p^{2(i-1)} + \sum_{i=1}^{n-1} c p^{2i-1}$. Let $O_T(\iota) = (T_1, T_2, \ldots, T_t)$.

9. Compute $T = \sum_{i=1}^{n} h(S_i) p^{2(i-1)} + \sum_{i=1}^{n-1} c p^{2i-1}$.

10. Randomly select a permutation σ_x from R_t, where $1 \le x \le t!$ and compute its inverse permutation $\sigma_y = \sigma_x^{-1}$.

11. Use Shamir's secret sharing to evaluate a polynomial modulo a prime $p' \ge \max(n, t!)$ and $|p'| = b$ with shadows $d_1, d_2, \ldots, d_n$ and constant term y.

12. For $1 \le i \le n$, Compute:

12.1. $c_i = e_{i1} \oplus e_{i2} \oplus \ldots \oplus e_{it} \oplus d_i$ and store in a public directory.

13. For $1 \le i \le n$, compute $O_{s_i}(\sigma_x)$ and send to the participant P_i along with S_i over a secure channel.

14. Compute $O_T(\sigma_x)$ and store it along with p and T in the public directory.

Notice that the inverse permutation σ_y is hidden in the second polynomial and hence the exact position of s_{it} in the t-tuple is not known to any participant as they don't know σ_x either, unless they pool all their shares together to evaluate the second polynomial.

5.3.2.2 Secret Recovery Phase

Without loss of generality assume that participants $P_1, P_2, \ldots, P_k$ decide to pool their shares.

1. Participants submit their first share S_i.

1.1. For participant P_i, compute $T' = \sum_{i=1}^{n} h(S_i)\, p^{2(i-1)}$.

1.2. For each P_i, check whether

$$\left\lfloor \frac{T - T'}{p^{i-1}} \right\rfloor \bmod p = 0$$

If the above equation does not hold, then P_i is a cheater.

1.3. If at least one cheater has been detected and identified then terminate the phase here.

2. For $1 \le j \le t$:

2.1. Participants submit their j th element of $O_{s_i}(\sigma_x)$.

2.2. For participant P_i, compute $T' = \sum_{i=1}^{n} h\left(s'_{ij}\right) p^{2(i-1)}$.

2.3. For each P_i, check whether

$$\left\lfloor \frac{T_j - T'}{p^{i-1}} \right\rfloor \bmod p = 0$$

If the above equation does not hold, then P_i is a cheater.

2.4. If at least one cheater has been detected and identified then terminate the phase here.

3. For $1 \le i \le k$:

3.1. Compute $d_i = e_{i1} \oplus e_{i2} \oplus \ldots \oplus e_{it} \oplus c_i$.

4. Use Langrange's polynomial interpolation to find x from these d_i's and retrieve the permutation σ_y from R_t through the unranking function.

5. For $1 \le i \le k$:

5.1. Apply the inverse permutation $\sigma_y = \sigma_x^{-1}$ to obtain $O_{s_i}\left(\sigma_x^{-1}\sigma_x\right) = O_{s_i}(\iota)$.

5.2. If $\lceil |p|/t \rceil \neq |p|/t$, remove $\lceil |p|/t \rceil t - |p|$ last bits from the first element of $O_{s_i}(\iota)$.

5.3. Compute $con\left(S_i, e_{i1}\right) = s_i = f(i)$.

6. Reconstruct the polynomial $f(x)$ from these k shares using Langrange's polynomial interpolation.

7. Recover the secret as $S = f(0)$.

At any stage of the recovery phase, if the cheaters change their elements in the t-tuples, the procedure will be terminated. And no one will submit the remaining elements of the t-tuples. It will be shown in the next section that the probability of successful cheating in this scheme depends upon the parameter t which can be adjusted to decrease this probability.

5.3.3 Security Analysis

The above scheme acts both as a cheating detection and prevention scheme. It can be shown that it prevents cheating with a probability of $(t-1)/t$. For a detailed security and complexity analysis, see [24].

5.3.4 The Key Safeguarding Protocol

With the Secret Sharing Scheme in our hand we can now illustrate our protocol. We use the notation $SecretShareGenerate(S,k,n)$ to denote the generation of n shares of the secret S with k being the threshold. Similarly, $SecretRecovery(S_1,S_2,\ldots,S_k)$ represents secret recovery from the k shares. Our secret sharing scheme as described above will automatically deal with false shares during the recovery. The Key safeguarding protocol is as follows:

Protocol 4: $KeySafeGuard(A,M)$

INPUT: A PseudoRandom Number Generator
OUTPUT: Shares S_1, S_2 and S_3

1. A generates an AES Key K_{safe}
2. A encrypts the important data using this key
3. A compute $SecretShareGenerate(K_{safe},2,3)$ to generate three shares S_1, S_2 and S_3.
4. A sends $E_{M_{pub}}(S_1)$ to M and stores one of the shares in the same device and the other in another device.

Table 4. Key Safeguarding Protocol.

After some time, A might need to read the data on its mobile device. It can do so by applying $SecretRecovery(S_1,S_2)$ or $SecretRecovery(S_2,S_3)$. However in the first case it has to authenticate with M again before it can recover the key.

5.4 Authorization Service

The Authorization Service is only the part of M and not A. This decides whether A is allowed to use M 's services and if yes then which of M 's services can it utilize. These access control decisions will be based on A and M 's policies and are handled by the Knowledge Management service in MAGI. This is indicated in the deployment diagram as an interface to the Knowledge Management Policy Manager. The authorization service makes decisions based on the feedback from the Policy Manager. Furthermore, if the Trust Manager is used, the Authorization Service can communicate with it to make stronger access control decisions.

5.5 Delegation Service

This service has the primary objective of securely delegating the job to grid services and ensuring "single sign on" as offered by GSI. The use of Elliptic Curve Cryptography (ECC) between A and M produces a severe limitation in the straight forward usage of RSA based X.509 certificates. If the grid services are capable of computing elliptic curve operations then the elliptic curves based proxy certificates can be easily used. However, most deployments of Grids use RSA based X.509 certificates. Thus the grid services cannot validate the ECDSA certificates. We solve this problem by using dual certificates. The CA issues two certificates to A. One is the RSA certificate and the other is the ECDSA certificate. After authenticating with M, A sends its RSA certificate along with the key pair to M. M can then use this certificate to interact with the grid services. However, after completing the job, M should no longer keep it with itself; the reason being that a failure or an attack on M may result in the loss of the key-pair along with the certificate. Furthermore A should also not keep this key in its mobile device due to fears of the device being stolen.

We again use the notion of secret sharing schemes to overcome this problem. After completing the job, M computes a fixed number of shares and sends them to the other instances of MAGI and destroys the original copy of the key. A destroys its copy as well. When A wants to send a job request later, it generates an ECC proxy certificate which shows that A has given M_1 the authority of doing the job on its behalf. M_1 can show this proxy certificate to any $k-1$ instances of MAGI and reconstruct the RSA private key. In this way we can ensure job delegation without using expensive RSA operations on the client side. The Delegation Protocol thus consists of an initial phase in which A sends both its RSA certificate and its key and then the consequent sessions can take place with A sending the proxy certificate to enable M to complete the job. After job completion, M destroys its copy of A's RSA key. The two protocols are described below:

Protocol 5: $DelegationInit(M_1)$

INPUT: A's RSA Private Key A_{RSA}, k and n

OUTPUT: n shares $S_1, S_2, \ldots, S_n$ of A_{RSA}

1. A sends its RSA certificate and A_{RSA} to M_1.
2. M_1 computes $SecretShareGenerate(A_{RSA}, k, n)$ to generate n shares $S_1, S_2, \ldots, S_n$.
3. M_1 sends these shares to $M_2, M_3, \ldots, M_n$ and keeps S_1
4. M_1 destroys $S_2, \ldots, S_n$ and A_{RSA}.

Table 5. Delegation Initialization Protocol.

Protocol 6: $DelegationReq(A, M_1)$

1. A generates an ECC proxy certificate P_{ECC}
2. A sends P_{ECC} to M_1.
3. M_1 uses P_{ECC} to validate with any $k-1$ MAGI instances $M_2, M_3, \ldots, M_k$
4. M computes $SecretRecovery(S_1, S_2, \ldots, S_k)$ to recover A's RSA secret key.
5. M uses this key and A's RSA certificate to communicate with the grid

Table 6. Delegation Protocol

5.6 Trust Manager

MAGI (M) will have to interact between large number of different mobile devices. Sometimes the users of these devices belong to different network domains governed by different security policies. At other times a new mobile device may want to interact without any prior history of interaction. MAGI would like to grant access privileges to these devices or allow a requested action based on some notion of trust. It would like to evaluate the requesting client on the basis of its past interactions with other Grid services and/or with other instances of MAGI. A global trust evaluation model becomes necessary in this situation enabling the communicating parties to determine the trust for each other. Even though the client is an authenticated user, it still might not cooperate well with MAGI and vice versa. We describe a trust model in the next section and describe a trust evaluation mechanism. Both A and M maintain the trust values about their past interactions.

5.7 Information Privacy Manager

To understand the role of the Information Privacy Manager, let us consider a scenario: There runs a medical diagnosis and information service on a Grid (say GMS), deployed to help the hospital(s) staff to pervasively carry out their duties. The medical service, upon request, processes and retrieves certain information about patients, analyzes it and returns some results. Suppose Dr. Alice has some mobile devices such as a laptop and a PDA. She would like to get some information about her patient by checking his medical history and getting a diagnosis from the GMS. She sends the request through her laptop to the middleware which delegates the job to GMS. After

job request she disconnects and continues on with her daily routine work. At a later time, she would like to get the information about her patient on her PDA. In particular she would be concerned only with some certain information (such as the Blood Pressure) of the patient and not all information returned by the medical service. The middleware has already retrieved the information from GMS and has temporarily stored it in its repository, waiting for the client to reconnect. Upon searching for the keyword "Blood Pressure", the middleware will return only the data categorized under this keyword and remove the remaining data from its repository. The data regarding the patient's medical history and his diagnosis is highly sensitive and the security policy of the hospital maintains that the data should not be accessible or presented to someone not related. The middleware is an access point to all the community in the hospital. Different doctors and other hospital staff request different jobs for different patients through it. Ideally the data residing on the intermediary (middleware) should be kept secret so that no one else should get the knowledge about the data. The key is that data should be private and searchable at the same time. The only parties who should know the data are Alice and the GMS. This naturally brings us to the notion of Public Key Encryption with Keyword Search (PEKS) discovered by Boneh et al [48]. They originally proposed the scheme in the scenario of an email gateway routing emails to user's different devices depending on the keywords in the email.

Suppose Dr. Alice has a PDA through which she would like to get some information about her patient by checking his medical history and getting a diagnosis from the GMS. She sends the request through her PDA to MAGI which delegates the job to GMS. After job request she disconnects and continues her other work. At a later time, she would like to get the information about her patient on her PDA. In particular she would be concerned only with some certain information and not all information returned by GMS to MAGI, such as the Blood Pressure of the patient. MAGI has already retrieved the information from GMS and has temporarily stored it in its repository. The information stored in MAGI is in the form of keywords and data, both of which are kept secret from MAGI. Our goal is to let Alice retrieve the information she desires without leaking any more information than necessary to MAGI. As MAGI is a gateway for access to the Grid, a number of different clients with different devices have access to it. Therefore sensitive unencrypted data is highly vulnerable to outside intruders.

Although this feature seems pretty important for the privacy of user's information residing on MAGI for a temporary period of time, the implementation of it, as we shall see, requires implementation of an IBE like system. If the grid side security has full support of this feature then it can be easily deployed. The current deployment of Grid does not support this mechanism. Hence we only describe it as an optional extension.

5.8 Performance Comparison of the Models

In the next simulation we calculated the time to completion of the job request, with Authentication and then with Encryption. Due to the lack of space we will not go into the details of the firing times of the transitions used in these two models. Figure 3, shows the difference in times to completion. As expected, even with the security features, the time to completion takes just a little more time. This shows the benefit of using Elliptic Curve Cryptography with AES. Both of them are fast and suitable for mobile devices.

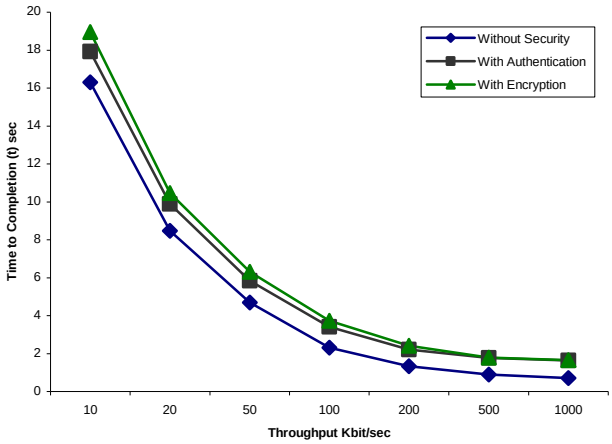

Figure 4. Time to Completion (t) of the three Petri Nets against changing wireless throughput

However, the relative difference between the times to completion, though being subtle at lower bandwidths, increases with increasing bandwidth, almost being two times at bandwidths of about 1000 Kbit/sec. This is due to the fact that at higher bandwidths the transmission costs are minimized and the performance difference is mostly due to the computational overhead.

Chapter 6: Grid Resource Scheduling

6.1 Introduction

This chapter introduces the Grid and related terms and explains the development of prototype Grid services. We also explain the tools and technologies used along with the required basic configuration steps.

6.1.1 The Grid

'The Grid' as a term in computing world, was formulated in the last decade. It referred to an envisioned advanced distributed computing paradigm with capabilities to ultimately assist in solving complex science and engineering problems beyond the scope of existing computing infrastructures [BLUEPRINT]. The concept has evolved considerably over these years. The growing popularity has also resulted in various kinds of 'grids', common ones being known as Data grids, Computational grids, Bio grids, Cluster grids, Science grids, among many others [FOSTERARTICLE]. Effort is in progress to converge the concepts related to the architecture, protocols, and applications of these grids to formulate a *single* paradigm – the Grid.

The article in [FOSTERARTICLE] lists various characteristics that may be significant in determining whether a distributed computing system meets the requirements to be a grid. According to [FOSTERARTICLE], such a system qualifies to be a grid, which:

a. coordinates resources that are not under centralized control.

b. utilizes standard, open, general-purpose protocols and interfaces.

c. promises to deliver non-trivial qualities of service.

Furthermore, these grids – if they follow common inter-grid protocols for authentication, authorization, access control, resource discovery, resource access, and resource sharing – will attempt to congregate into the Grid.

The grid community is working towards this goal, and standardized protocols are expected in near future, that may revolutionize the computing world; as Internet Protocol (IP) did with the Internet.

6.1.2 Globus

The Globus grid-middleware is provided by the Globus project [GLOBUS-PAGE] bundled as a set of tools called Globus Toolkit. The toolkit has 3 components known as *pillars*. These are:

- Resource Management
- Information Services
- Data Management

The Globus Toolkit uses the GSI (Globus Security Infrastructure) to provide a common security protocol for each of the pillars. GSI is based on public key encryption, X.509 certificates and Secure Socket layer (SSL) protocol.

6.1.2.1 GRAM (Globus Resource Allocation Manager)

The Globus Resource Allocation Manager provides a standard interface to all the local resource management tools a site uses. The Globus resource management has the high-level global resource management services layered on top of local resource-allocation services [GLOBUS-PAGE]. The GRAM service is provided by a combination of the ***gatekeeper*** and the ***jobmanager***. The gatekeeper performs the task of authenticating an inbound request using GSI, and mapping the user's global ID on the grid to a local username. The incoming request specifies a specific local service to be launched, the latter usually being a jobmanager. The user needs to compose the request in a Resource Specification Language (RSL) that is handed over to the jobmanager by the gatekeeper. After parsing the RSL, the jobmanager translates it into the local scheduler's language. The GRAM also provides the capability to stage in executables or data files, using Global Access to Secondary Storage (GASS). The jobmanager contacts the client before the job submission for retrieval of the staged in files.

6.1.2.2 MDS (Metacomputing and Directory Service)

The Globus Metacomputing and Directory Service provides an LDAP based information infrastructure, suited for grid environments. There are 2 components of MDS implementation – ***GRIS*** and ***GIIS***. The Grid Resource Information Service (GRIS) implements a uniform means to query resources on a grid for current status

and configuration. The Grid Index Information Service (GIIS) component of MDS provides a framework to form an index over various GRIS's or other GIIS's. This combines the information of an entire system, thereby giving a method to explore a coherent system image. Both these components [GLOBUS-PAGE] are currently implemented using the *slapd* server provided by OpenLDAP and follow the Lightweight Directory Access Protocol (LDAP). MDS utilizes the concept of *information providers* – software programs that act as probe utilities or sensors to the smaller components of a grid. Since it is based on LDAP, MDS needs a schema to be built in that represents the hierarchy and rules of the information to be retrieved from the components of a grid. Caching mechanisms are used to make the retention of data more efficient, based on the time-sensitivity of a piece of information.

6.2 Installing Support Software

This topic lets you know which support software you must download as well as other software you may want.

- Java SDK
- Ant
- Junit
- C compiler
- YACC (or Bison)
- GNU tar
- Tomcat
- .NET
- JDBC-compliant Database (RFT, RLS)

6.2.1 Java SDK

Since the underlying code of GT3 is written in Java, you must install the Java platform on any machines running the Toolkit.

Required for: GT3 Webservices components

Recommended Versions: 1.3.1 through 1.4.x (1.3.1 may be incompatible with certain services - see Important Notes)

Download Link: http://java.sun.com/j2se

JAAS library is required as a separate download if you are using JDK 1.3.1. Some higher level services like the Index Service and Execution Services use Xindice, which is not compatible with version 1.3.1 on some platforms. Please see the Xindice FAQ for more details.

JDK 3.2.0 has bug 1596 that restricts the Java GRAM to using version 1.4.x. This will be fixed in an update package. If you don't need to build any source, you may use JRE instead.

If you are using Sun's JDK, the safest version to use is JDK 1.4.2. Version 1.4.1 sometimes hangs during compilation for unknown reasons.The MMJFS component (part of GRAM) might not properly work with the IBM JDK on Linux.

Follow either Sun or IBM's instructions for installing JDK. The current distribution as of this writing is 1.4.2.

Make sure to set the JAVA_HOME environment to the installation directory and add $JAVA_HOME/bin to your PATH environment.

6.2.2 Ant

Ant is a Java-based build tool required for the GT3 installation. The toolkit is composed of many Java classes that need to be combined or built to form a functioning program. Ant is used to execute an Ant-based build script that automates the build process (if the toolkit needs to be re-buit at any point, Ant can skip actions that have already been completed.)

Required for: Webservices installation

Recommended Version: 1.6.1

Download Link: http://jakarta.apache.org/ant

You can continue to use Jakarta Ant 1.4.1 if you replace crimson.jar in your $ANT_HOME/lib directory with the xerces.jar that comes with our distribution.

Follow Jakarta's instructions for installing Ant 1.6.x. The current distribution as of this writing is 1.6.1.

Make sure to set the ANT_HOME environment to the installation directory and add $ANT_HOME/bin to your PATH environment.

6.2.3 Junit

Junit is a Java-based testing framework that facilitates regression tests. If you run tests from source, the junit.jar class must be included with Ant for the GT3 installation.

Required for: Gridservices (however it is optional if you are only installing the GT3 Core component)

Recommended Version: 3.8.1

Download Link: http://www.junit.org

Follow Junit's instructions for installing Junit 3.8.1. Copy junit.jar to the $ANT_HOME/lib or put it on the $CLASSPATH.

6.2.4 C compiler

Required for: Any C code. The only components that do not require this are the GT3 core component and the Higher-level GARs.

Recommended Version: Anything except gcc 3.2 (the RedHat 8 default compiler). Gcc 3.2 triggers bug 488, our most-reported bug.

Download Link: http://www.gnu.org/software/gcc/ for GCC. Vendor compilers also work.

6.2.5 YACC (or Bison)

Required for: building C bindings and the ogsi-find-service-data client from source.

Recommended Version: the latest version

Download Link: http://www.gnu.org/software/bison/bison.html

Your compiler probably already has this installed. Otherwise, you can download the GNU version. For installing YACC (or Bison), follow the instructions from the website.

6.2.6 GNU Tar

Required for: Unpackaging software

Recommended Version: The latest version

Download Link: http://mirrors.kernel.org/gnu/tar/

If you are running Linux, your system comes with GNU Tar. If you are running solaris, /usr/bin/tar is not GNU Tar and you will need to install it. For installing GNU Tar, follow the instructions from the INSTALL file inside the file download.

6.2.7 Jakarta Tomcat

Used for: your web service container. This is optional because the GT3 installation provides a standalone web service container for testing purposes.

Recommended Version: 4.1.24 (4.0.6 has also been tested to work)

Download Link: http://jakarta.apache.org/tomcat

For installing Tomcat, follow the instructions from the website.

6.2.8 Java Database Connectivity (JDBC) compliant Database

JDBC is an API for Java that allows access to a wide range of SQL databases. JDBC is similar to the open standard API Open Database Connectivity (ODBC), which is aligned with The Open Group.

Used for: the Reliable File Transfer (RFT) service and Replica Location Service (RLS), which require a database backend. For licensing reasons, we ship the PostgreSQL JDBC driver, but other JDBC compliant databases should be compatible.

Recommended Version: not applicable

Download Link: http://www.postgresql.org

6.2.8.1 Installing a JDBC compliant database:

In our prototype, we'll use PostgreSQL. PostgreSQL consists of 11 RPMs, but only three of them are required to install a functioning database. This procedure installs all of the RPMs and edits two configuration files: (Note: This installation creates the user postgres. This user will be used during the creation of the database.)

Download Postgresql from http://www.postgresql.org/ and install RPMs.

Allow clients to connect via TCP/IP-based connections:

a) As root, edit /etc/init.d/postgresql.

b) Find the line that starts with the postmaster command.

c) Add the -i flag after the -o flag.

d) Remove -p ${PGPORT}.

Tell the database to listen for requests on the host machine by modifying the host-based authentication file, pg_hba.conf.

a) As root, edit /var/lib/pgsql/data/pg_hba.conf.

b) Find the host stanza (or record) and enter it with the IP address for any users from any machines that should have access to the database (the default value is 127.0.0.1)

c) Remove the # character from the beginning of the line.

After saving the file, start the database by running:

/etc/init.d/postgresql start

6.2.9 Installing Globus toolkit

As a security precaution, we recommend installing GT3 as a non-root user. GT3 is designed to run as few components at elevated privileges as possible. As you will see in the Configuration section, there are two files that will be made setuid. One of these needs to be setuid root in order to run a User Hosting Environment as another user. The other needs to be setuid to an account which owns the host certificate. That can either be a separate non-root user, or you could choose to have that be root also. You don't need to make the certificate owner choice right now; it will be covered in the Configuration section.

6.2.9.1 Installing the Globus Toolkit 3.2 - Binary Installers

a) If you plan to run the WS GRAM service we recommend that you install as the user that will run the WS GRAM service. Go to the Download page and choose the binary installer you want to install. You will have to choose between the all/base/preogsi. For the linux installers, run rpm -q glibc to choose between the glibc2.2/glibc2.3 choices. If the first two digits are "2.3", like "glibc-2.3.2-27.9", then you would use the glibc-2.3 installer. If the first two digits are "2.2", like "glibc-2.2.93-5", then you would use the glibc-2.2 installer.

These binaries are only known to work on the distribution used to generate them (RH7.3 for the glibc2.2, and Fedora Core 1 for glibc2.3). We have had bug reports even on machines which report the same Major.Minor.Point release of glibc, but are from a different distribution. If you experience GridFTP errors with the binary installation, please rebuild from source.

b) As globus, untar the binary installer.

c) Make sure that ANT_HOME and JAVA_HOME are set, and that ant and java are on your PATH. If you are using JDK 1.3.1, make sure JAAS is in your CLASSPATH before you run the installer.

d) Run:

./install-gt3-bin /path/to/install

Some of the resource management configuration can use the packages under the schedulers/ directory. You may not want to delete the installer directory until after you are done configuring.

6.3 Configuring Globus Toolkit

Step 1:

As globus, set GLOBUS_LOCATION to where you installed the Globus Toolkit.

This will either be

export GLOBUS_LOCATION=/path/to/install

or

setenv GLOBUS_LOCATION /path/to/install.

Step 2:

Source $GLOBUS_LOCATION/etc/globus-user-env.{sh,csh} depending on shell.

.sh for Bourne shell

.csh for C shell

Certificate Authority (CA) options

Your best option is to use an already existing CA. You may have access to one from the company you work for, or an organization you are affiliated with. Some universities provide certificates for their members and affiliates. Contact your support organization for details about how to acquire a certificate. You may find your CA listed in the TERENA Repository.

If you do not have an existing CA, you can set up a CA for your own use with the Globus SimpleCA package. SimpleCA provides a wrapper around the openssl CA functionality and is sufficient for simple Grid services. Alternatively, you can use

openssl's CA.sh

command on its own.

You can also use an online certificate service. However, this option should only be used as a last resort because it does not fulfill some of the duties of a real Certificate Authority.

If you must use this option, please see the following link for instructions:

http://gcs.globus.org:8080/gcs.

If you do not have access to an existing CA and want to use SimpleCA, continue with step 3. If you already have a CA, you will need to follow their configuration directions. If they include a CA setup package, you may continue to step 11 directly. If they do not, you will need to create an /etc/grid-security/certificates directory and include the CA cert and signing policy in that directory. See Configuring a Trusted CA for more details. Then proceed to step 11.

SimpleCA:

Creating users

Step 3:

Make sure you have the following users on your machine:

Your user account, which will be used to run the client programs.A generic globus account, which will be used to perform administrative tasks such as starting and stopping the container, deploying services, etc. This user will also be in charge of managing the SimpleCA. To do this, make sure this account has read and write permissions in the $GLOBUS_LOCATION directory.

SimpleCA: Running the setup script

A script was installed to set up a new SimpleCA. You only need to run this script once per grid.

Step 4:

Run the setup script:

$GLOBUS_LOCATION/setup/globus/setup-simple-ca

Subject name: This script prompts you for information about the CA you wish to create:

The unique subject name for this CA is:

cn=Globus Simple CA, ou=simpleCA-mayed.mcs.anl.gov, ou=GlobusTest, o=Grid

Do you want to keep this as the CA subject (y/n) [y]:

The common name (cn) is Globus Simple CA , which identifies this particular certificateas the CA certificate within the GloubusTest/simpleCA-hostname domain. The organizational unit (ou) is GlobusTest , and the second ou is specific to your hostname. That identifies this CA from other CAs created by SimpleCA by other people.

The organization is Grid.

Step 5:

Press y to keep the default subject name (recommended).

Email: The next prompt looks like:

Enter the email of the CA (this is the email where certificate

requests will be sent to be signed by the CA):

Step 6:

Enter the email address where you intend to receive certificate requests. It should be your real email address that you check, not the address of the globus user.

Expiration: Then you'll see:

The CA certificate has an expiration date. Keep in mind that once the CA certificate has expired, all the certificates signed by that CA become invalid. A CA should regenerate the CA certificate and start re-issuing ca-setup packages before the actual CA certificate expires. This can be done by re-running this setup script. Enter the number of DAYS the CA certificate should last before it expires. [default: 5 years (1825 days)]:

This is the number of days for which the CA certificate is valid. Once this time expires, the CA certificate will have to be recreated, and all of its certificates regranted.

Step 7:

Accept the default (recommended).

Passphrase:

Generating a 1024 bit RSA private key

.....++++++

.....++++++

writing new private key to '/home/globus/.globus/simpleCA//private/cakey.pem'

Enter PEM pass phrase:

The passphrase of the CA certificate will be used only when signing certificates (with grid-cert-sign). It should be hard to guess, as its compromise may compromise all the certificates signed by the CA.

Step 8:

Enter your passphrase.

Important: Your passphrase must not contain any spaces.

Finally you'll see the following:

A self-signed certificate has been generated for the Certificate Authority with the subject:

/O=Grid/OU=GlobusTest/OU=simpleCA-mayed.mcs.anl.gov/CN=Globus Simple CA

If this is invalid, rerun this script

setup/globus/setup-simple-ca

and enter the appropriate fields.

The private key of the CA is stored in /home/globus/.globus/simpleCA//private/cak

ey.pem

The public CA certificate is stored in /home/globus/.globus/simpleCA//cacert.pem

The distribution package built for this CA is stored in /home/globus/.globus/simpleCA//globus_simple_ca_68ea3306_setup-0.17.tar.gz

This information will be important for setting up other machines in your grid. The number 68ea3306 in the last line is known as your CA hash. It will be some 8 hexadecimal digit string.

Step 9:

Press any key to acknowledge this screen.

Your CA setup package finishes installing and ends the procedure with the following reminder:

****
***

Note: To complete setup of the GSI software you need to run the

following script as root to configure your security configuration

directory:

/opt/gt3/setup/globus_simple_ca_68ea3306_setup/setup-gsi

For further information on using the setup-gsi script, use the -help

option. The -default option sets this security configuration to be

the default, and -nonroot can be used on systems where root access is

not available.

****
***

setup-ssl-utils: Complete

We'll cover this last step in the next section. For now, just notice that it refers to your $GLOBUS_LOCATION and the CA Hash from the last message.

Step 10:

To finish the setup of GSI, run as root (or, if no root privileges are available, add the -

nonroot option to the command line):

$GLOBUS_LOCATION/setup/globus_simple_ca_CA_Hash_setup/setup-gsi -default

The output should look like:

setup-gsi: Configuring GSI security

Installing /etc/grid-security/certificates//grid-security.conf.CA_Hash...

Running grid-security-config...

Installing Globus CA certificate into trusted CA certificate directory...

Installing Globus CA signing policy into trusted CA certificate directory...

setup-gsi: Complete

Requesting and signing host certificates

You must request and sign a host certificate and then copy it into the appropriate directory for secure services. The certificate must be for a machine which has a consistent name in DNS; you should not run it on a computer using DHCP where a different name could be assigned to your computer.

Step 11:

Request a host certificate: As root, run:

grid-cert-request -host 'hostname'

This creates the following files:

/etc/grid-security/hostkey.pem

/etc/grid-security/hostcert_request.pem

(an empty) /etc/grid-security/hostcert.pem

Note: If you are using your own CA, follow their instructions about creating a hostcert (one which has a commonName (CN) of your hostname), then place the cert and key in the /etc/grid-security/ location. You may then proceed to user certificates.

Step 12:

Sign the host certificate: as globus, run:

grid-ca-sign -in hostcert_request.pem -out hostsigned.pem

A signed host certificate, named hostsigned.pem is written to the current directory.

When prompted for a passphrase, enter the one you specified in step 8 (for the private key of the CA certificate.)

Step 13:

As root, move the signed host certificate to /etc/grid-security/hostcert.pem. The certificate should be owned by root, and read-only for other users. The key should be read-only by root. Requesting and signing user certificates

Users also must request user certificates, which you will sign using the globus user.

Step 14:

Request a user certificate: As your normal user account (not globus), run:

grid-cert-request

After you enter a passphrase, this creates

~$USER/.globus/usercert.pem (empty)

~$USER/.globus/userkey.pem

~$USER/.globus/usercert_request.pem

Email the usercert_request.pem file to the SimpleCA maintainer.

Note: If you are using your own CA, follow their instructions about creating a usercert (one which has a commonName (CN) of your real name), then place the cert and key in the ~USER/.globus/ location. You may then proceed to verifying proxy creation.

Step 15:

Sign the user certificate: as the SimpleCA owner globus, run:

grid-ca-sign -in usercert_request.pem -out signed.pem

When prompted for a password, enter the one you specified in step 8 (for the private key of the CA certificate.)

Now send the signed copy (signed.pem) back to the user who requested the certificate.

Step 16:

As your normal user account (not globus), copy the signed user certificate into

~/.globus/ and rename it as usercert.pem, thus replacing the empty file.

The certificate should be owned by the user, and read-only for other users.

The key should be read-only by the owner

Step 17:

To test that the SimpleCA certificate is installed in /etc/grid-security/certificates and that your certificate is in place with the correct permissions, run:

user$ grid-proxy-init -debug -verify

After entering your passphrase, successful output looks like:

[bacon@mayed schedulers]$ grid-proxy-init -debug -verify

User Cert File: /home/user/.globus/usercert.pem

User Key File: /home/user/.globus/userkey.pem

Trusted CA Cert Dir: /etc/grid-security/certificates

Output File: /tmp/x509up_u1817

Your identity: /O=Grid/OU=GlobusTest/OU=simpleCAmayed.

mcs.anl.gov/OU=mcs.anl.gov/CN=User Name

Enter GRID pass phrase for this identity:

Creating proxy .....++++++++++++

.....++++++++++++

Done

Proxy Verify OK

Your proxy is valid until: Sat Mar 20 03:01:46 2004

Change the ownership and access permissions

Run the setperms.sh script to change the ownership of some Globus files under the

$GLOBUS_LOCATION/bin directory. This step allows resource management tools to run as root.

Step 18:

As root, run:

$GLOBUS_LOCATION/bin/setperms.sh

Add Authorization

Add authorizations for users:

Step 19:

Create /etc/grid-security/grid-mapfile as root.

You need two pieces of information - the subject name of a user, and the account name it should map to.

The syntax is one line per user, with the certificate subject followed by the user account name.

Run grid-cert-info to get your subject name, and whoami to get the account name:

bacon$ grid-cert-info -subject

/O=Grid/OU=GlobusTest/OU=simpleCAmayed.

mcs.anl.gov/OU=mcs.anl.gov/CN=Charles Bacon

bacon$ whoami

bacon

The corresponding line in the grid-mapfile:

"/O=Grid/OU=GlobusTest/OU=simpleCAmayed.

mcs.anl.gov/OU=mcs.anl.gov/CN=Charles Bacon" bacon

The quotes around the subject name are important, because it contains spaces.

Step 20:

At this step, you have a single machine configured. Recall that in Step 8 a CA setup package was created in .globus/simpleCA//globus_simple_ca_HASH_setup-0.17.tar.gz. If you want to use your certificates on another machine, you will have to install that CA setup package on that machine. To install it, copy that package to the second machine and

run:

$GLOBUS_LOCATION/sbin/gpt-build globus_simple_ca_HASH_setup-0.17.tar.gz

gcc32dbg

Then you will have to perform the setup-gsi -default from step 10. If you are going to run services on the second host, it will need a host certificate and a grid-mapfile also. You may re-use your user certificates on the new host. You will need to copy the requests to the host where the SimpleCA was first installed in order to sign them.

Now you are ready to use secure Grid services.

Chapter 7: Related Work

7.1 Mobile-to-Grid Middleware

Various efforts have been made to solve the problem of mobile-to-Grid middleware. Signal [15] proposes a mobile proxy-based architecture that can execute jobs submitted to mobile devices, so in-effect making a grid of mobile devices. A proxy interacts with the Globus Toolkit's Monitoring and Discovery Service to communicate resource availability in the nodes it represents. The proxy server and mobile device communicate via SOAP and authenticate each other via the generic security service (GSS) API. The proxy server analyzes code and checks for resource allocation through the monitoring and discovery service (MDS). After the proxy server determines resource availability, the adaptation middleware layer component in the server sends the job request to remote locations. Because of this distributed and remote execution, the mobile device consumes very little power and uses bandwidth effectively. Also their efforts are more inclined towards QoS issues such as management of allocated resources, support for QoS guarantees at application, middleware and network layer and support of resource and service discoveries based on QoS properties.

In [16] a mobile agent paradigm is used to develop a middleware to allow mobile users' access to the Grid and it focus's on providing this access transparently and keeping the mobile host connected to the service. Though they have to improve upon the system's security, fault tolerance and QoS, their architecture is sufficiently scalable. GridBlocks [17] builds a Grid application framework with standardized interfaces facilitating the creation of end user services. They advocate the use of propriety protocol communication protocol and state that SOAP usage on mobile devices maybe 2-3 times slower as compared to a proprietary protocol. For security, they are inclined towards the MIDP specification version 2 which includes security features on Transport layer.

7.2 Trust Model

Since mid '90s the research community has outlined the key role of trust management models to develop more complex and dependable computer systems. From this, the importance of trust model was first highlighted by Blaze *et al* in their seminal paper [3]. Subsequently, Josang [7] presented an interesting classification of trust relationships and its implication to traditional security concepts.

Until now, several trust models have been proposed in the literature for different distributed systems [8]. For the Grid scenario, X.509[9] and SPKI[10] seem adequate which propose a central Certificate Authority (CA) based trust model. However, there are a number of issues related to proxy/delegation certificates that are serious drawbacks of these models. A two-level trust model for Grid based on graph topology was proposed in [11]. They use different trust evaluation metrics for centralized grid domains and distributed Virtual Organizations (VO). A peer recommended trust model was proposed in [12] for ubiquitous computing systems. Their trust management scheme through recommendation lacks certain aspects such as the weighted recommendation of peers based on their prior interactions. In [13], a decentralized trust and reputation model for multi agent systems has been proposed whereas a probabilistic trust model is proposed in [14] for mobile agents. Both these models lack a fundamental requirement, i.e., very old recommendations should not be relevant in predicting the behavior of an entity. Another probabilistic trust model called the Beta Reputation System (BRS) [15] works by giving ratings about other users in the system. All these trust models can be generally categorized into probabilistic models and others in which the trust evaluation formulae are tuned to give the desired result.

In the fields of Ubiquitous Computing, research has paid much attention to build autonomous trust management as fundamental building block to design the future security framework. Up to now, research has focused mainly on the propagation and composition of trust information [16,17,18,19] while paying less attention to how direct trust information is actually built. Though focused on distributed trust computation, [20,21] face the problem of building trust from past experience. Michiardi *et al* [22] proposed an organic reputation –based framework to enforce collaboration in ad-hoc networks. Peer reputation is built by evaluation a mix of directly collected information, undirected feedback, and eventually multiple interaction classes.

Chapter 8: Conclusion

In this paper we identified the potential of enabling mobile devices access to the Grid. We focused on providing solutions related to distributed computing in wireless environments, particularly when mobile devices intend to interact with grid services. An architecture for a middleware layer is presented which facilitates implicit interaction of mobile devices with grid services. This middleware is based on the web services communication paradigm. It handles secure communication between the client and the middleware service, provides software support for offline processing, manages the presentation of results to heterogeneous devices (i.e. considering the device specification) and deals with the delegation of job requests from the client to the Grid. We also demonstrated that the addition of such a middleware causes minimum overhead and the benefits obtained by it outweigh this overhead.

In future we intend to provide multi-protocol support in order to extend the same facilities to devices that are unable to process SOAP messages. Moreover, we will continue to focus on handling security, improving support for offline processing and presentation of results depending upon the device. Along with this implementation we intend to continue validating our approach by experimental results.

References

1. A. Puliafito, S. Riccobene, M. Scarpa, "Which paradigm should I use?: An analytical comparison of the client-server, remote evaluation and mobile agents paradigms'", *IEEE Concurrency and Computation: Practice & Experience*, vol. 13, pp. 71-94, 2001.
2. A. Bobbio, A. Puliafito, M. Telek, "A modeling framework to implement preemption policies in non-Markovian SPNs". *IEEE Transactions on Software Engineering*, vol. 26, pp. 36-54, Jan. 2000.
3. M. Telek, A. Bobbio, "Markov regenerative stochastic Petri nets with age type general transitions". *Application and Theory of Petri Nets, 16th International Conference* (*Lecture Notes in Computer Science* 935). Springer-Verlag, pp. 471–489, 1995.
4. A. Bobbio, A. Puliafito, M. Scarpa, M. Telek, "WebSPN: A WEB-accessible Petri Net Tool". *International Conference on WEB based Modeling and Simulation*, San Diego, California, pp. 137–142, 11–14 January 1998.
5. WebSPN 3.2: ing-inf.unime.it/webspn/
6. W. Hoschek, Web service discovery processing steps. http://www-itg.lbl.gov/~hoschek/publications/icwi2002.pdf
7. UDDI specification, www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
8. SOAP Framework: W3C Simple Object Access Protocol ver 1.1, World Wide Web Consortium recommendation, 8 May 2000; www.w3.org/TR/SOAP/
9. GT3 GRAM Architecture, www-unix.globus.org/developer/gram-architecture.html
10. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001
11. V. Welch, F. Siebenlist, I. Foster, et al., Security for grid services. HPDC, 2003.
12. Von Welch, Ian Foster, Carl Kesselman, et al., X.509 Proxy Certificates for dynamic delegation. Proceedings of the 3rd Annual PKI R&D Workshop, 2004.
13. Vipul Gupta, Sumit Gupta, et al., Performance Analysis of Elliptic Curve Cryptography for SSL. Proceedings of ACM Workshop on Wireless Security - WiSe 2002 pages 87-94, Atlanta, GA, USA, September 2002, ACM Press.
14. Della-Libera Giovanni, D.B., Hondo Maryann et al., Security in a Web Services World; A Proposed Architecture and Roadmap, 2002, International Business Machines and Microsoft Corporation. A joint security whitepaper from IBM Corporation and Microsoft Corporation. April 7, 2002, Version 1.0
15. J. Hwang, P. Aravamudham Middleware Services for P2P Computing in Wireless Grid Networks. IEEE Internet Computing vol. 8, no. 4, July/August 2004, pp. 40-46
16. D. Bruneo, M. Scarpa, A. Zaia, A. Puliafito, Communication Paradigms for Mobile Grid Users. Proceedings 10th IEEE International Symposium in High-Performance Distributed Computing, (2001)
17. Gridblocks project (CERN) http://gridblocks.sourceforge.net/docs.htm

18. I. Foster, C. Kesselman, and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, Int'l J. Supercomputer Applications, vol. 15, no. 3, 2001, pp.200-222.
19. The Java WSDP Registry Server - http://java.sun.com/webservices/docs/1.0/tutorial/doc/RegistryServer.html
20. Java API for XML Registries - http://java.sun.com/webservices/docs/1.0/tutorial/doc/JAXR.html
21. P. Maes. Concepts and experiments in computational reflection. In 2nd Conference on Object Oriented Programming Systems, Languages and Applications, pages 147–156.
22. A. Fuggetta. Understanding code mobility. In Transactions on Software Engineering, volume 24, pages 342–361. IEEE.
23. Sun-Microsystems. Java. http://java.sun.com/j2se/.
24. G. Shepperd, M. Kadoda. Comparing software prediction techniques using simulation. In IEEE Transactions on Software Engineering, volume 27, pages 1014–1022. IEEE, 2001.
25. H. Weinsberg, Y. Israel. A programming model and system support for disconnected-aware applications on resource-constrained devices. In 24th International Conference on Software Engineering, pages 374–384, 2002.
26. U. Kalim, H. Jameel, A. Sajjad, Mobile-to-grid middleware: An approach for breaching the divide between mobile and grid environments. In 4th International Conference on Networking, pages 1–8. Springer Verlag, 2005.
27. M. Marija. Improving availability in large, distributed, component-based systems via redeployment. Technical Report USC-CSE-2003-515, Center for Software Engineering, University of Southern California, 2003.
28. M. Kistler, J. Satyanarayanan. Disconnected operation in the coda file system. In 13th ACM symposium on Operating Systems Principles, pages 213–225. ACM, 1991.
29. J. Howard. An overview of the andrew file system. In USENIX Conference, pages 213–216, 1988.
30. M. Noble, B. Satyanarayanan. Agile application-aware adaptation for mobility. In 16th ACM Symposium on Operating Systems Principles. ACM, 1997.
31. R. P. T. Popek, G. Guy. The ficus distributed file system: Replication via stackable layers. In Technical Report CSD- 900009. University of California, 1990.
32. A. Joseph. Rover: A toolkit for mobile information access. In 15th ACM Symposium on Operating Systems Principles, pages 156–171. ACM, 1995.
33. S. B. G. Conan, D. Chabridon. Disconnected operations in mobile environments. In 16th International Parallel and Distributed Processing Symposium, page 118, 2002.
34. S.P. Marsh. Formalising Trust as Computational Concepts. Ph.D Thesis, University of Stirling, 1994
35. C. English, P. Nixon, S. Terzis, A. McGettrick and H. Lowe. Dynamic Trust Model for Ubiquitous Computing Environment. In Proceeding of the Workshop on Security in Ubiquitous Computing UBICOMP 2002.
36. M. Blaze, J. Feigenbaum, and J. Lacy: "Decentralized trust management". In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pp.164-173, May 1996.
37. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis: "The KeyNote Trust Management System - Version 2". Internet Engineering Task Force, September 1999. RFC 2704.

38. Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss: "REFEREE: Trust Management for Web Applications," World Wide Web Journal, 2 (1997), pp. 706--734.
39. Hung Q. Ngo, Anjum Shehzad, Saad Liaquat Kiani, Maria Riaz, Kim Anh Ngoc, Sungyoung Lee, “Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework”, The 2004 International Conference on Embedded & Ubiquitous Computing (EUC2004), Springer-Verlag Lecture Notes in Computer Science, Japen, August 26-28 2004.
    A. Josang. The right type of trust for distributed systems. In New security paradigms workshop, Lake Arrowhead (CA, USA), pages 119–131, September 1996.
40. S.D. Ramchurn, D. Hunyh, and N.R. Jennings. “Trust in multi-agent systems”. Knowledge Engineering Review, 19(1), 2004.
41. Mendes, S. and Huitema. A new approach to the X.509 framework: Allowing a global authentication infrastructure without a global trust model. In Proceedings of NDSS’95
42. C Ellison et al. Spki certificate theory. September 1999. Internet Request for Comments: 2693.
43. Tie-Yan Li, Huafei Zhu, Kwok-Yan Lam: A Novel Two-Level Trust Model for Grid. ICICS 2003: 214-225.
44. M. Carbone, M. Nielsen and V. Sassone. A Formal Model for Trust in Dynamic Networks. Proceedings of IEEE International Conference on Software Engineering and Formal Methods (SEFM '03), 2003.
45. D. Huynh, N. R. Jennings, and N. R. Shadbolt. Developing an Integrated Trust and Reputation Model for Open Multi-Agent Systems. In Proceedings of 7th International Workshopon Trust in Agent Societies, pages 66–74, New York, July 2004. ACM Press.
46. Jigar Patel et al. “ A Probabilistic Trust Model for Handling Inaccurate Reputation Sources”. AAMAS 2005.
47. R. Ismail and A. Josang. “The Beta Reputation System”. In Proceedings of the 15th Bled Conference on Electronic Commerce, Bled, Slovenia, 2002.
48. M. Richardson, R. Agrawal, and P. Domingos. Trust management for the semantic web. In International Semantic Web Conference, Sanibel Island (FL, USA), pages 351–368, October 2003.
49. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In International Conference on World Wide Web, Budapest (Hungary), pages 640–651, May 2003.
50. G. Theodorakopoulos and J. S. Baras. Trust evaluation in ad-hoc networks. In ACM Workshop on Wireless security, Philadelphia (PA, USA), pages 1–10, October 2004.
51. R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of trust and distrust. In International Conference on World Wide Web, New York (NY, USA), pages 403–412, May 2004.
52. P. Michiardi and R. Molva. CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad-hoc networks. In IFIP Communica-

tion and Multimedia Security Conference, Portoroz (Slovenia), pages 107–121, September 2002.
53. S. Ganeriwal and M. B. Srivastava. Reputation-based framework for high integrity sensor networks. In ACM Workshop on Security of ad-hoc and sensor networks,Washington (DC, USA), pages 66–77, October 2004.
54. P. Michiardi and R. Molva. CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad-hoc networks. In IFIP Communication and Multimedia Security Conference, Portoroz (Slovenia), pages 107–121,September 2002.
55. ITU-T Recommendation X.509 (2000 E). Information Technology. Open systems interconnection-The Directory: Public-key and attribute certificate frameworks.
56. S.R. White, J.E. Hanson, I. Whalley, D.M. Chess and J.O. Kephart. An Architectural Approach to Autonomic Computing. Proc. of 1st International Conference on Autonomic Computing, 2004
57. K. Herrmann, G. Mühl, and K. Geihs, "Self-Management: The Solution to Complexity or Just Another Problem?" IEEE Distributed Systems Online, vol. 6, no. 1, 2005
58. M. Abrams, C. Phanouriou, et. al., UIML: An Appliance-Independent XML User Interface Language, WWW8 / Computer Networks, 1999, http://www.oasis-open.org
59. GridBlocks: Helsinki Institute of Physics, (CERN). http://gridblocks.sourceforge.net/docs.htm
60. F. Hupfeld. "Log-Structured Storage for Efficient Weakly-Connected Replication". In Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS) Workshops, 2004
61. Bruneo, D., Scarpa, M., Zaia, A., Puliafito, A.: Communication Paradigms for Mobile Grid Users. Proceedings 10th IEEE International Symposium in High-Performance Distributed Computing (2001)
62. Hwang, J., Aravamudham, P.: Middleware Services for P2P Computing in Wireless Grid Networks. IEEE Internet Computing vol. 8, no. 4 (2004) 40-46
63. I Foster, C Kesselman, J Nick, S Tuecke, The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, Open Grid Service Infrastructure WG, Global Grid Forum, June, 2002