Design Of A Peer-to-peer-based Scalable Grid Service

A Job Meta-Scheduling service

V L A D I M I R M A R I N K O V I C

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LECS-2004-59

Design Of
A Peer-to-peer-based
Scalable Grid Service

A Job Meta-Scheduling Service

By

Vladimir Marinkovic

Examiner:
Vladimir Vlassov
Institute of Microelectronics and
information technology, IMIT
The Royal Institute Of Technology

Industrial Supervisor:
Konstantin Popov
Swedish Institute of Computer Science
(SICS)

The Royal Institute of Technology

Stockholm, October 2004

Abstract

Grid technology is evolving. Open Grid Service Architecture (OGSA) defines grid services based on Web services technology. The grids are starting to expand towards another popular resource-sharing technology: the peer-to-peer overlay networks. As grids are evolving, issues of scalability need to be addressed. P2P is a very scalable resource sharing technology. These two technologies are converging towards each other.

The project includes study and a survey of Grid services (in the context of OGSA: Open Grid Service Architecture), technologies for Web services, P2P overlay networks, and related work. As a case study, the thesis presents design of a job meta-scheduling Grid service. The service is based on the peer-to-peer (P2P) technology, namely, on the DKS (Distributed K-ary search System) that is a structured P2P system. The service incorporates the properties of the system it is based on, such as decentralization, fault-tolerance, self-organization and scalability.

Sammanfattning

Grid-tekniken utvecklas. Open Grid Service Architecture (OGSA) definierar Grid services med hjälp av Web services-teknik. Långsamt börjar gridar också expandera mot en annan trendig teknik för resursdelning: peer-to-peer nätverk. När gridar växer måste frågor kring skalbarhet lösas. P2P-nätverk är en väldigt skalbar teknik för delning av resurser. Dessa tekniker konvergerar långsamt mot varandra.

Projektet inkluderar litteraturstudie och genomgång av dessa tre tekniker Gridar, Web service och P2P-nätverk. Som fallstudie presenterar rapporten en design av en tjänst för schemaläggning av jobb i gridar. Den är baserad på P2P-teknik, närmare bestämt på DKS (Distributed K-ary search System), som är ett strukturerat P2P-system. Tjänsten får egenskaper av systemet den baseras på, såsom decentralisering, feltolerans, självorganisation och skalbarhet.

Table of Contents

Table of figures

1 Introduction

The grid technology provides an infrastructure for coordinated resource sharing in multi-institutional, virtual organizations. The grid communities are wishing to create an infrastructure that will allow creation of a widespread,global collaboration network. People should be able to join and share resources with other members of that network. The amount of effort for joining the network, either for a resource sharing or a service consumption, should be minimized.

If this is to be achieved, the technology should be based on open standards for service invocation, service look-up, etc. The standards must be extensible, to be able to support further development and evolution of the technology. The Web services are open-standard, extensible infrastructure for service description, invocation and look-up. They are text-based, i.e. XML-based, and are attractive for use within grid systems. In recent years efforts were made to adopt this technology into the grids.

Current implementations of the grids are quite closed, mostly centralized sharing environments. In spite of that, the grid systems are getting bigger. The virtual organizations are growing in size. Researchers are expecting the virtual organizations to start introducing the economical component in the grids, i.e. start renting the computational power, the storage space and the network bandwidth. This will further lower the bounds for membership in a grid system, which will grow even more. With the growth, the technology will have to address issues of scalability and ease of deployment.

The peer-to-peer overlay network is a scalable technology, which have similar goals of distributed resource sharing. An overlay network is a network that runs on top of another network, such as a TCP/IP network. The overlay contains the connectivity information, closest peers, etc. The great scalability of these systems is very attractive for the grid environments. Because of the common goal, researchers are starting to look into whether the technology of P2P systems can be adopted into the grid environments.

1.1 Goals And Expected Results

The main purpose of this Master thesis project is two-fold:

1. Study of related work on Grid and Web services, (structured) overlay networks with DHT functionality and their use in P2P applications;
2. Design of at least one of Grid services (such as meta-scheduling service) based on the overlay network infrastructure called the DKS system developed at KTH and SICS. The main features of the Grid service to be achieved are good scalability and low-cost self-organization.

We expect that design of a Grid service on the structured overlay network with DHT functionality will help to evaluate whether the DHTs are useful and convenient (easy to use) for Grid services, as well as help to evaluate other properties of the structured network that might be useful for Grid services.

Expected results of this project include (but not limited to):

1. A survey of the three technologies: Grids, Web services and P2P overlay networks; and related work towards the implementation of OGSA.
2. An architecture (structure, interfaces, algorithms and protocols) of a Grid service as a peer-to-peer application based on an overlay network with DHT functionality.

1.2 Structure Of The Thesis

The thesis is structured as follows. Chapter 2 presents the survey of studied technologies. It is divided into three sections, each covering a studied technology: Peer-to-peer (P2P) Computing, Web Services and Grid Services. The design of a peer-to-peer-based meta-scheduling grid service is presented in Chapter 3. In chapter 4, some implementation issues are presented and clarified. The conclusions in the project are summarized in chapter 5. The appendix includes a listing of a service definition as well as a listing of a Java interface.

2 Background

2.1 Peer-To-Peer Computing (P2P)

Peer-to-peer (P2P) computing is still a young technology and still evolves. Many researchers are trying to define P2P computing.

Shirky in [3] defines the P2P like this:

> *"P2P is a class of applications that takes advantage of resources -- storage, cycles, content, human presence -- available at the edges of the Internet. Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers."*

Milojicic in [1] defines the term P2P as:

> *"The term 'peer-to-peer' (P2P) refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner."*

What is common in all definitions is that some terms are reoccurring, such as: decentralized, distributed, resources and sharing.

2.1.1 Architecture

The goal with P2P systems is to share distributed resources between equivalent nodes (peers). Different P2P systems have taken different approaches to solving this problem (see section 2.1.2). All these approaches must solve some infrastructural problems, such as maintenance of connections, group management, routing of messages, etc, and on top of that infrastructure, build some resource sharing services or applications.

Milojicic in [1] talks about an informal P2P system architecture, where the components of an abstract P2P system are presented. The architecture is depicted in Figure 1. The ordering of the components does not strictly follow the layering.

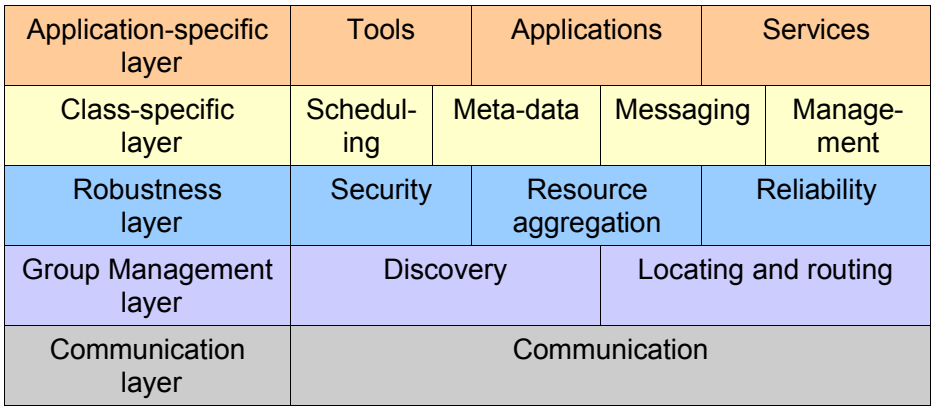

***Figure 1** – An Informal System Architecture*

The communication component is responsible for maintaining an application-level connection to other peers in a dynamic environment of a P2P system. The component for discovery is responsible for discovering other nodes (peers) in a P2P system. The component for locating and routing is responsible for locating distributed resources in the network and for routing of messages. This is where major differences exist between different approaches to P2P computing.

Security, in a distributed manner, is one of the most difficult things to achieve in P2P systems. Since nodes are acting both as clients and servers, peers that wish to access resources must authenticate. This is often solved by centralizing the issues of security.

The resource aggregation is concerned with aggregation of resources. The reliability component in P2P systems is concerned with reliable behavior of the system. This is often solved by taking advantage of redundancy, for example: restarting of computation, resending of messages, replication of data.

The components in class-specific layer represent different classes of P2P applications. The scheduling applies to compute-intensive applications and the meta-data applies to content and file management applications. The messaging applies to collaborative applications and the management component is concerned with the management of the underlying P2P infrastructure.

In the Application layer the components, the tools, the applications and the services, implement the functionality of a P2P application, such as the distributed scheduling, the file sharing, the collaborative applications, such as the chatting and the messaging.

2.1.2 Approaches

There are different approaches to, or models of, the peer-to-peer computing. A couple of different classifications of P2P systems exists. The systems can be divided into three generations of P2P systems, i.e. according to steps of evolution. In an another classification, the models of the P2P systems can be divided into:

- Centralized model
- Flooding model
- Document routing model
- DHTs

Actually the Document routing model and DHTs are based on the same idea and can be classified into the same group. Here, they are divided because the DHT model is of great importance for future work in the project.

Centralized Model

This model was implemented by the pioneering P2P applications such as Napster [21], [22]. In this model, the peers connect to a centralized directory where they publish a list of files that they are sharing. The search queries are sent to this centralized directory, which then searches for the best match. Download of a file is executed by direct access between peers.

Another view of this model is that there exist two services offered by this network: a directory service and a storage service. The directory service is centralized and the storage service is distributed [2].

The centralized organization of this model has the disadvantage of having a centralized repository that is queried by every peer. This can result in some limits in scalability, if the number of peers or the requests increases. Another disadvantage is that there exists a single point of failure in this model. If the centralized directory is to fail, the whole network will stop functioning. There also exists some centralized infrastructure that must be maintained [1]. Since all requests are sent to the directory, the requests are not propagated through the network. This will give a low traffic of messages in the network.

Flooding Model

In this model peers use the flooding algorithm to communicate with other peers. This model is used by Gnutella [23]. To be able to participate in a network, a peer has to know an IP-address of one peer, from which it learns about the other peers by flooding. The peers are directly connected to some small number of peers in the network. The discovery of the shared resources, in this model, is done by flooding a request to all directly connected peers, which, in their turn, flood the request to their directly connected peers, etc. The request is propagated through the network. The propagation of the request is limited by a Time-To-Live value. Peers that receive the request and do share the requested resource will answer directly to the requesting peer.

This flooding approach is pure P2P, i.e. has no centralized component and is completely distributed, but does not scale well. The intensive message passing demands high network bandwidths. In limited communities it can be an efficient algorithm. An effort was made by P2P communities (e.g. Kazaa [20], Gnutella [23]) to limit the demand on bandwidth by introducing “super-peers”. These “super-peers” act like directory services, and reduce flooding. Caching of recent requests is another way of reducing the message passing within the network.

Document Routing Model

Another model is the document routing model, used by FreeNet [26]. Peers in a P2P systems of this model are assigned a random identification number. Each peer knows about some predefined number of other peers. When a document is published, a checksum is calculated based on its content and filename. The document is then routed towards a peer with most similar ID number. If the current peer has the most similar ID number, then the document is stored. The requests for a document are routed in the same manner. Every peer that receives a document during the routing will keep a local copy of it.

This model is very efficient in large, global communities. The down side of this model is that a document's checksum must be known in advance. The other problem that can occur in these networks is the islanding problem. The network can split in parts with no link to each other. A P2P systems implementing this model can't have high data location guarantees.

Distributed Hash Tables (DHTs)

These systems, such as Chord [27], Pastry [24], Tapastry [25], CAN, DKS, are based on the idea of document routing, but are trying to achieve an abstraction of distributed hash tables (DHTs). The primary goal of these algorithms is to reduce the number of hops when locating a resource in the network and to reduce the size of routing tables. An efficient DHT allows balanced distribution of data among the nodes and a logarithmic-time lookup.

A peer, or a node, is assigned an identifying number based on a cryptographic hash of some system attribute, such as the IP-address. The peers are then joined into a network in some algorithm-specific, structured manner. In Chord, a circular identifier space is used, while in CAN a d-dimensional space. Pastry/Tapastry uses a mash.

For storing data (making resources available) into a DHT, the key-data pairs are used. A key for a data item is obtained through hashing. Both keys and peer identifiers are hashed into the same identifier space. The key-data pairs are stored at nodes according to the given structure. The structured topology of the network makes locating data (resources) a routing problem. The routing tables are of a logarithmic size in Chord, Pastry and Tapestry. The CAN algorithm has a fixed number (d) of entries in the routing tables. Also there exists a maximum path length in such structured networks. Therefore, the DHTs can have high look-up guarantees.

Distributed K-ary Search System (DKS)

The DKS system is a structured peer-to-peer overlay network. It implements the DHT functionality. The DKS is based on Chord and is well described by the above description of the DHTs. It uses a virtual k-ary spanning tree of hight

$$\log_k(N),$$

where N is number of nodes in the network. The lookup is resolved by following a path of the spanning tree. The Chord uses a binary spanning tree. This ensures logarithmic lookup path length. The DKS organizes the peers in a circular identifier space and has the routing tables of logarithmic size.

In the DKS the identifier space is larger then the actual number of the participating nodes. Every node is responsible for some interval of the identifier space. When a data is stored into the DKS, the data is forwarded towards the node which

is responsible for identifier given by the hashed key of that data. Since the keys and the identifiers are both hashed into the same identifier spare, the storing of the data is a matter of the routing. When data arrives to the node responsible for the identifier given by the key, it is stored there. The lookup is the opposite operation. Given a key, there is a node which is responsible for that part of the identifier space. The request for data-retrieval is forwarded to that node.

Besides the DHT functionality, the DKS system provides other services, such as efficient broadcast and multicast of messages. For a detailed description of the DKS, see [31],[32],[33].

2.2 Web Services

When the Web services technology emerged, big companies, such as IBM, Microsoft, Sun, etc, have realized the potential power of it in the e-Business. They have driven the development and the evolution of this technology. This has resulted in a fast research and development of the Web services. The standardization of the de facto standard protocols is done by The World Wide Web Consortium (W3C).

IBM in [5] defines the Web services to be:

> *"A technology that allows applications to communicate with each other in a platform- and programming language-independent manner. A Web service is a software interface that describes a collection of operations that can be accessed over the network through standardized XML messaging. It uses protocols based on the XML language to describe an operation to execute or data to exchange with another Web service. A group of Web services interacting together in this manner defines a particular Web service application in a Service-Oriented Architecture (SOA)."*

W3C in [6] defines a the Web service like this:

> *"[Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]"*

The Web services provide an interface for distributed applications to communicate. It defines a set of protocols that enables applications to publish, search, provide and consume a service. All protocols are based on XML, which makes the Web services platform- and programming language-independent.

Conceptually, there are three different roles in a typical Web service communications:

- Service Provider – A platform that provides a service
- Service Consumer – An application that searches and consumes a service
- Service Registry – A searchable registry of services. It has knowledge of where a service provider is and how a service is consumed.

A typical Web service communication is depicted in Figure 2. When a service provider wishes to publish a service it will create a service description using the Web Services Description Language (WSDL) protocol. This description tells a service consumer what a request for service invocation should contain and how a response from the service provider will contain. (For more detail WSDL see section 2.2.3)

This description, together with other information on how the service provider should be contacted, is published at a service registry. When a service consumer wishes to consume a service, it first searches the service registry for an appropriate service and receives a service description. Having the service description the

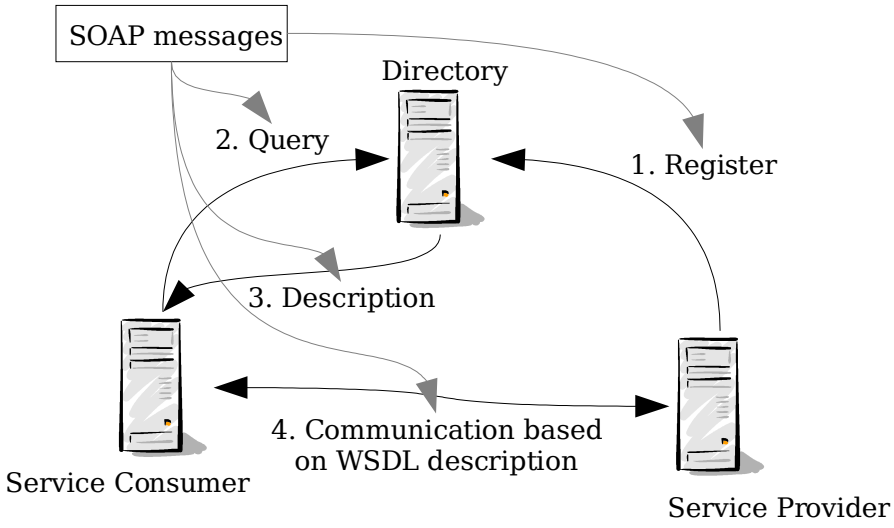

Figure 2 *– A Typical Web Service Setup*

consumer knows how to invoke a service. The entire message passing between these three roles is done over a messaging protocol called the SOAP.

2.2.1 Messaging Protocol – SOAP

SOAP is a lightweight, XML based protocol used for exchanging messages. It is stateless and one-way messaging protocol. By combining a SOAP messages and underlying protocols or application-specific data, it can be used to achieve more complex communications (e.g. request/response, conversations, etc). SOAP relies on underlying protocols for network access (see Figure 3). The SOAP messages can be used in combination with, or enveloped in, a variety of protocols, such as HTTP, SMTP, FTP, etc.

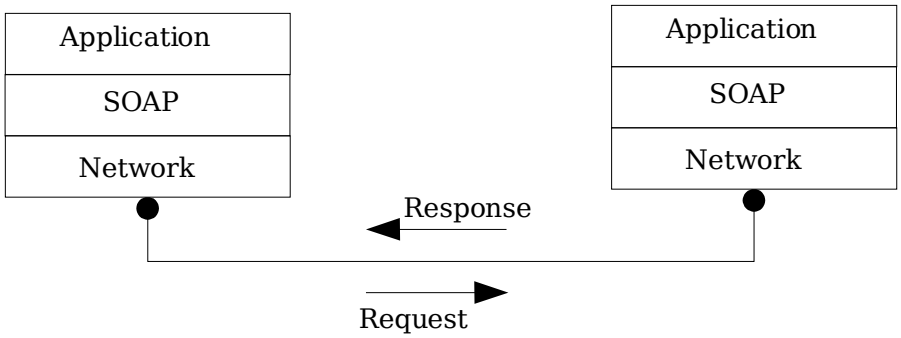

Figure 3 *– Communication With SOAP Messages*

SOAP provides an envelope for sending structured data from a SOAP sender to a SOAP receiver. The SOAP envelope consists of two parts: a SOAP header and a SOAP body.

The SOAP header is optional. The header is used for passing non-application-specific data from a sender to a receiver. This data can be directives or some contextual information for processing of the message. Subsequent blocks of the header are called header blocks.

The SOAP body contains an application payload. The contents of the SOAP body of a message are purely application-specific, and are not part of the SOAP specification [8].

2.2.2 Web Services Description Protocol - WSDL

Web Services Description Language (WSDL) is an XML format for describing a Web Service and how they should be bound to a network address. A service is modeled as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. Web services are defined by using the following six major elements: *Types, Message, Port Type, Binding, Port* and *Service*. These elements can be classified into [9]:

- The service interface definition (*Binding, PortType, Message* and *Types*)
- The service implementation definition (*Service* and *Port*)

The service interface definition contains reusable elements and is a reusable service definition that can be used by many service implementation definitions. The *PortType* element defines the operations of Web service. Operations describe actions for the messages supported by a Web service. WSDL has four operations that an endpoint can support [7]:

- One-way. A message received – no response required
- Request-response - Request received - send a response
- Solicit-response - A request for a response
- Notification - A message sent

Input and output parameters of an operation are defined by the Message element. The *Types* element describes complex types that are used within a message. The *Binding* element defines a message format and protocol details for operations and messages defined by a particular *portType*.

The service implementation definition describes how a service interface is implemented by a service provider. A service is modeled by the *service* element, which can contain several *Port* elements. The *Port* element associates a binding from interface definition to an endpoint, URL.

These two definitions might be divided into two separate documents. This is because of the re-usability of the interface definitions. An interface may be defined according to some industrial standards and implemented in many services at many companies. This is not a requirement, though. All six within a single document may define a Web service, as well.

When a service consumer finds a service it wishes to consume, the description of the service must be processed. The description contains enough information to

generate SOAP messages, which are to be sent to a Web service, as well as to decode a reply-message from the service.

WSDL is the minimum standard service description that is necessary for correct invocation of Web services. WSDL defines how a service is consumed. Additional descriptions are necessary to fully describe a service, such as in what context is the service relevant [9].

2.2.3 Service Discovery And Publishing

A service consumer must have the service description to be able to consume a service. The Web service must publish this information, so that it is accessible by the consumer. The simplest scenario is with statically linked services. This means that at the design time, a developer have located a Web service that is to be consumed by the application, retrieved a description of the service and made it available to the application on the local (or remote, but accessible) file system.

A more complex scenario is that service is not known at the design time. In this case the Web service must be located and the description retrieved at run-time. For this purpose, some kind of repository is necessary. Universal Description and Integration (UDDI) is a powerful, searchable directory for Web service publication.

UDDI was originally developed by uddi.org. Uddi.org was comprised by the technology and business leading companies in an effort to enable companies and individuals to quickly and easily find and use Web services. Eventually, UDDI was transferred to OASIS. It is not part of the standardization effort done by W3C.

UDDI provides a definition of a set of services supporting the description and discovery of [9]:

1. businesses, organizations, and other Web Services providers,
2. the Web Services they make available, and
3. the technical interfaces, which may be used to access those services.

UDDI is based on a common set of standards, including HTTP, XML, XML Schema, and SOAP.

The UDDI registry is a logically centralized directory. Physically it is a distributed service with multiple root nodes. Nodes replicate data with each other on a regularly. When a business registers with one instance of the registry service, the data is automatically replicated to other root nodes. Then, it is freely available for anyone who wishes to invoke provided services.

There are four different types of UDDI nodes [9]:

- Internal Enterprise Application UDDI node – Node for Web services that will be accessed only by internal enterprise applications
- Portal UDDI node – Node for Web services that will be accessed by business partners
- Partner Catalog UDDI node – Node for Web services that will be accessed only by particular company
- E-marketplace UDDI node – Node for publicly available services

In UDDI, Web services are organized by businesses. The entries contain basic information on businesses, detailed business data, information about company's business properties. All information must be provided at time of publication of a service. The information can be added to the UDDI registry via Web site. There are also tools, which exploit programmatic service interfaces, to register with the UDDI directory.

The directory supports mechanism for finding Web services based on type of interface, the binding information, properties, taxonomy of the service, business information, etc.

2.3 Grid Services

The term "the Grid" first appeared in mid 1990s in a proposal for distributed computing infrastructure for advanced science and engineering [7].

IBM defines grid computing in [14]:

> *"Grid computing enables the virtualisation of distributed computing and data resources such as processing, network bandwidth, and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. At its core, grid computing is based on an open set of standards and protocols that enable communication across heterogeneous, geographically dispersed environments."*

As by the Globus Alliance in [15], grids are

> *"The Grid refers to an infrastructure that enables the integrated, collaborative use of high-end computers, networks, databases, and scientific instruments owned and managed by multiple organizations."*

The basic problem grid computing is trying to solve is defined by Ian Foster in [11]:

> *"The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional, virtual organizations."*

The resources in this context have broad meaning, such as computational power, network bandwidth, storage space, etc. A virtual organization (VO) is a set of individuals or physical organizations that share the resources according to predefined set of rules [11].

The vision of grid computing environment is often described as analogous to a power grid [28]. When an appliance is plugged in the user of the power grid can receive the electrical power, without bothering from where and how this power is delivered. A local utility store will provide an interface through which the electrical power grid can be accessed. The infrastructure will provide a virtual generator. This virtual generator consists of many different power sources. This grid is very reliable and adopts to consumers demands.

A grid environment can be described in same way. When the basic infrastructure is installed, the user will be able to access a virtual computer through an appropriate interface. This virtual computer is reliable and adopts according to the consumer's demands. The virtual computer consists of a variety of different resources, which are not visible to the user, just as power sources are not visible to consumer of electrical power.

The grid computing is quite new technology and has not fully evolved. To be able to reach the above-mentioned vision, a reliable and secure infrastructure must exist. This infrastructure must be built upon general standards and syntaxes, such as Open Grid Services Architecture (OGSA).

2.3.1 Virtual Organizations

The basic idea behind grids is resource sharing between physical organizations, to enable access to computational and storage power, collaboration and number of accessible instrumentation, etc. A group of individuals or physical organizations that share resources between each other according to some predefined rules are called virtual organization (VO). When a VO is created, representatives of physical organizations must meet, formally establish a VO, agree upon and define policies, describe contributions and responsibilities. After that, administrative privileges are assigned to some entity. This administrative entity will then assign privileges to all other entities within the VO. All participants must install appropriate middleware and expose their shared resources to the middleware. The users can then access and use resource within the limitation of assigned rights.

The sharing relationships within the VO are very dynamic. They can vary over time of day and in resources involved. Access to a resource can be allowed to some user groups and not to others. All these rules depend on the agreement, which is established at the time of the creation of the VO.

The concept of virtual organizations enables groups of organizations to share resources in some controlled fashion. This allows participants to collaborate and achieve common goals.

2.3.2 Grid Architecture

The architecture of grids can be divided in layers, as described in [11]. The components of the architecture are structured into layers. The components within same layer share characteristics. The layering of the grid architecture follows the hourglass model of layering. There are five layers in the architecture, which is depicted in Figure 4:

- Fabric
- Connectivity
- Resources
- Collective
- Application

The Fabric layer provides resources, which are made accessible through grid protocols. The resources can be physical (such as sensors, measurement equipment, etc) as well as logical (such as computer clusters, distributed file systems, etc).

The components in the Fabric layer implement resource-specific operations. At minimum, the resources must implement enquiry mechanisms for state, structure and capabilities, as well as resource management mechanisms, which provide control of quality of service.

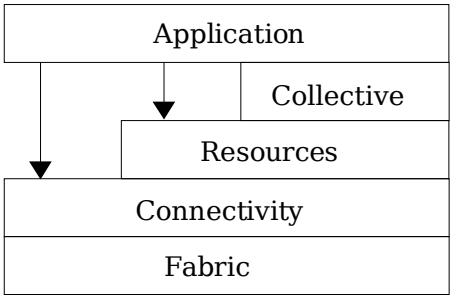

Figure 4 *– Grid Architecture*

The connectivity layer defines communication and authentication protocols required for network transactions between resources. Communication requirements include transport, routing and naming. Authentication should have the following properties:

- Single sign-on
- Delegation
- Integration with local security solutions
- User-based trust relationships

The resource layer builds on communication layer and provides protocols for secure negotiation, initiation, monitoring, control, accounting and payment. There are two classes of protocols in this layer:

- Information protocols - to query the state of a resource
- Management protocols - to negotiate access to a resource

The collective layer in the architecture provides services and protocols, which coordinate multiple resources. Protocols and services in this layer provide a wide range of sharing scenarios, such as:

- Directory services
- Co-allocation, scheduling and brokering services
- Workload management systems and collaboration frameworks
- etc

The application layer is where grid applications are implemented. Applications may use any of the protocols and services defined in any of the layers. They should be designed in terms of services.

2.3.3 OGSA And OGSI

In recent years, the grid community (the Global Grid Forum [16] and the Globus Alliance [15]) have made a lots of efforts to make the grid system architecture based on web services concepts and technologies. The results of these efforts are the Open Grid Service Architecture (OGSA) and Open Grid Service Infrastructure (OGSI).

Key advantages of Web services technologies that make them attractive to be employed for Grid Services include interpretability based on open text-based standards, modularity and ability for incremental implementation.

OGSA defines the architecture of a grid, including the infrastructure and the programming model for grid services. It introduces the notion of services to the grid environment, and focuses on the services that are provided, rather then physical (or logical) resources that are shared. OGSI defines the infrastructure that is required to achieve the properties of grid services defined in OGSA.

OGSA defines a layered architecture of grids. The layering consists of four layers (see Figure 5) [17]:

- Resources - physical resources and logical resources
- Web services, plus the OGSI extensions that define grid services
- OGSA architected services
- Grid applications

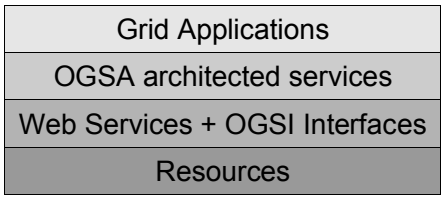

Figure 5 *– Grid Architecture*

The resources layer represents the shared resources in the grid. Resources can be physical or logical. Web services layer is the second layer in the OGSA architecture. Together with OGSI, it defines grid services, the basic infrastructure. OGSA architected grid services layer implements grid services, such as program execution, data services, and core services. Grid applications layer is where grid applications are implemented. These applications consume services offered within a grid.

To be able to implement a grid service there must exist some infrastructure that addresses grid services requirements. First, the grid environment can be very dy-

namic. A state of resources, sharing policies, dispatched work, system state, etc, may change and services may appear and disappear. The basic infrastructure must be able to handle creation, destruction and life cycle of services. Second, grid services have state. They can have attributes and data associated with them. This is something Web services can't handle. The OGSI specification defines grid services, which are built on top of Web services. It extends the definition of Web services (WSDL in particular) to provide dynamic, stateful and manageable Web services that are able to model grid resources.

The extensions, which OGSI is contributing to the web services layer, consist of five interfaces. The following are the interfaces defined in OGSI [13]:

- GridService
- Factory
- Notification
- ServiceGroup
- HandleResolver

Among these, the most important is the GridService. A grid service must implement this interface. It is the basic interface in OGSI. The behavior encapsulated by the GridService interface is that of querying and updating and managing the termination of the instance. Grid services that implement the ServiceGroup interface are grid services that maintain information about a group of other grid services.

A factory is used by a client to create a grid service instance. A client invokes a create operation on a factory and receives as response an identifier for the newly created service instance. The newly created grid service instance should be registered with a handle resolution service. The Factory interface must extend the GridService interface.

A grid service that implements the HandleResolver interface is called a handle resolver. When a grid service is instantiated by a factory, an identifier is returned. This identity is composed of two parts, a Grid Service Handle (GSH) and a Grid Service Reference (GSR). The HandleMap interface provides the means to obtain a GSR given a GSH. (For more detail on GSH and GSR see [12]).

A grid services' state changes as systems runs. Many interactions between services require notification of changing state. Grid services support an interface to permit other grid services to subscribe to changes.

2.3.4 Globus Toolkit And GRAM

The Globus Toolkit (GT) is developed by Globus Alliance [15]. It is an implementation of the Open Grid Services Infrastructure (OGSI) and provides a set of software components, that can be used either independently or together to develop higher-order services and/or applications. These components provide functionality for security, communication, information infrastructure, data management, resource management, fault detection, and portability. Some of the the core components are:

- The Globus Resource Allocation and Management (GRAM) provides resource allocation and process creation, monitoring, and management services.
- The Grid Security Infrastructure (GSI) provides a single-sign-on, run-anywhere authentication service.
- The Indexing service (generally called information service) provides information about available resources and services.

All necessary APIs (Application Programming Interfaces) and the command line utilities are provided with the software. For further information on the GT, see [15].

Grid Resource Allocation And Management (GRAM)

The Grid Resource Allocation and Management (GRAM) is a set of service components that provide a single standard interface for requesting and using remote resources for job execution. This interface allows clients to access a large variety of resources through one interface, and vice versa, allows the resources to communicate with clients through a single interface. In the hourglass model, depicted in Figure 6, the GRAM is neck of the hourglass. Above the GRAM there are applications and higher-order services, such as meta-schedulers and resource brokers. Below the GRAM are local control and access mechanisms.

Figure 6 *– The Hourglass Model Of GRAM*

The GRAM allows a client to submit, monitor and shutdown jobs remotely. There are four basic services that are provided by the GRAM (for detailed information, see [29].):

- (Master) Managed Job Factory Service (MJFS)
- Managed Job Service (MJS)
- File Stream Factory Service (FSFS)
- File Stream Service (FSS)

All jobs are executed on local systems as local users. The Grid Security Infrastructure (GSI) is used to authenticate both users and resources. It provides a mechanism for mapping a GSI identity to a local user account. This way a job can be executed as a local user.

The specification of resources for execution of a job is written in the Resource Specification Language (RSL). It is the XML-based language and a lot of power of the GRAM lies in it. With the attributes defined in RSL, such as executable, arguments, directories, execution times, etc, requirements for a job execution can be described in detail. If the attributes are not efficient, then the RSL allows extending the original set of attributes.

There are two containers associated with the GRAM, the Master Hosting Environment (MHE) and the User Hosting Environment (UHE). The MHE contains components for message redirection and instantiation of new UHEs. The Master

Managed Job Factory Service (MMJFS) is responsible for configuration of the Redirector component (see Figure 7; 1).

A client that wishes to execute a job must first instantiate a Managed Job Service (MJS). This is done by invoking the *createService* operation (see Figure 7; 2) of an instance of the Managed Job Factory Service (MJFS) in the UHE. When an operation-call arrives to the Redirector it has to find out what UHE this message should be forwarded to. It asks the Starter UHE component, which authenticates the user and maps it to the local user (see Figure 7; 3). If the UHE is not up and running (see Figure 7; 6), the Launch UHE component is contacted (see Figure 7; 4). It will launch an UHE for the user (see Figure 7; 5). The URL of the UHE is forwarded back to the Redirector. The operation-call is then forwarded to the MJFS service in the UHE. All subsequent messaging from this client will be redirected to this UHE, immediately (see Figure 7; 8).

The operation *createService* is invoked on the MJFS in the UHE (see Figure 7; 7). The result of this operation is a new instance of MJS service which will execute the request and schedule the job in the local scheduling system. To start the execution itself, the *start* operation must be invoked on the newly created MJS.

If a file staging, or a redirection of standard input and output is necessary, the File Stream Factory Service (FSFS) is used to instantiate the File Stream Service (FSS) which is then used to stream files, standard output or standard input to the given location.

The GRAM does not provide scheduling or resource brokering capabilities, neither does it provide accounting and billing features. It is assumed that these features are supplied by the local management mechanisms such as a queuing system or a scheduler.

Global meta-schedulers aren't developed by Globus, and should be provided by third-party providers. The main task of a meta-scheduler is to assist users (i.e. clients) in choosing an instance of GRAM, according to some previously decided algorithm, on which a job will execute. The simplest algorithm for choosing an instance is Round-Robin, i.e. equally spread jobs among existing instances. More advanced algorithms might be developed, so that the load of a node is considered.

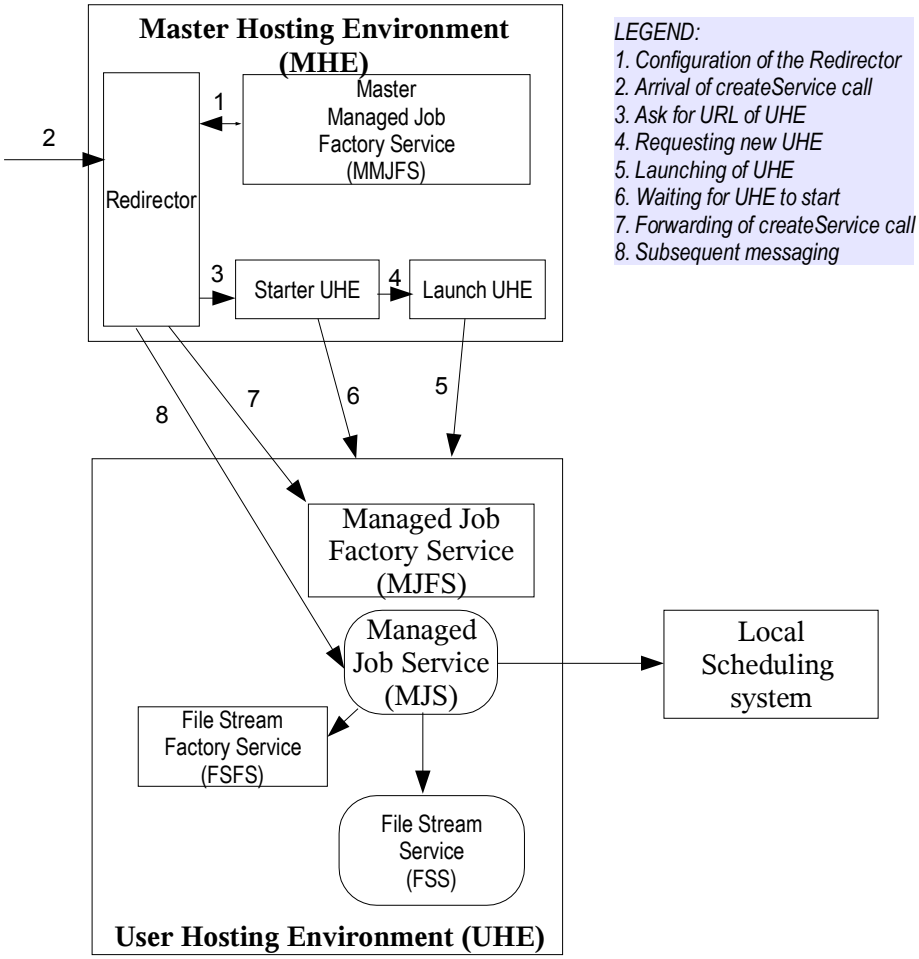

***Figure 7** – A Typical Communication In GRAM*

Information Services

In a grid, services contain some service data that defines their state. There could exist several instances of same service in the grid, and they can be distinguished by their state. This service data is represented in a standardized way, through the Service Data Elements (SDEs). The information services, also called Management and Discovery Services (MDS), is a broad framework that includes any part of a grid that generates, registers, indexes, aggregates, subscribes, monitors, queries, or displays the service data. Information services are implemented as Indexing Service and are one of the base services in the Globus Toolkit. [30]

Querying of different service data provides information about the resources. This information can be used for discovery of the resources or selection or optimization. This is important for design of applications and higher-order services. A wide variety of information about resources can be queried. For example, from

the Computing Element (CE), one could get information about a load on a particular processor, an architecture of a processor (an instruction set), policies on a particular host, a file system on a particular host, etc. The GLUE schema of the CE in the indexing service is shown in Figure 8. The figure depicts a schema of the available information about the resources (hosts). This information can be retrieved from the CE.

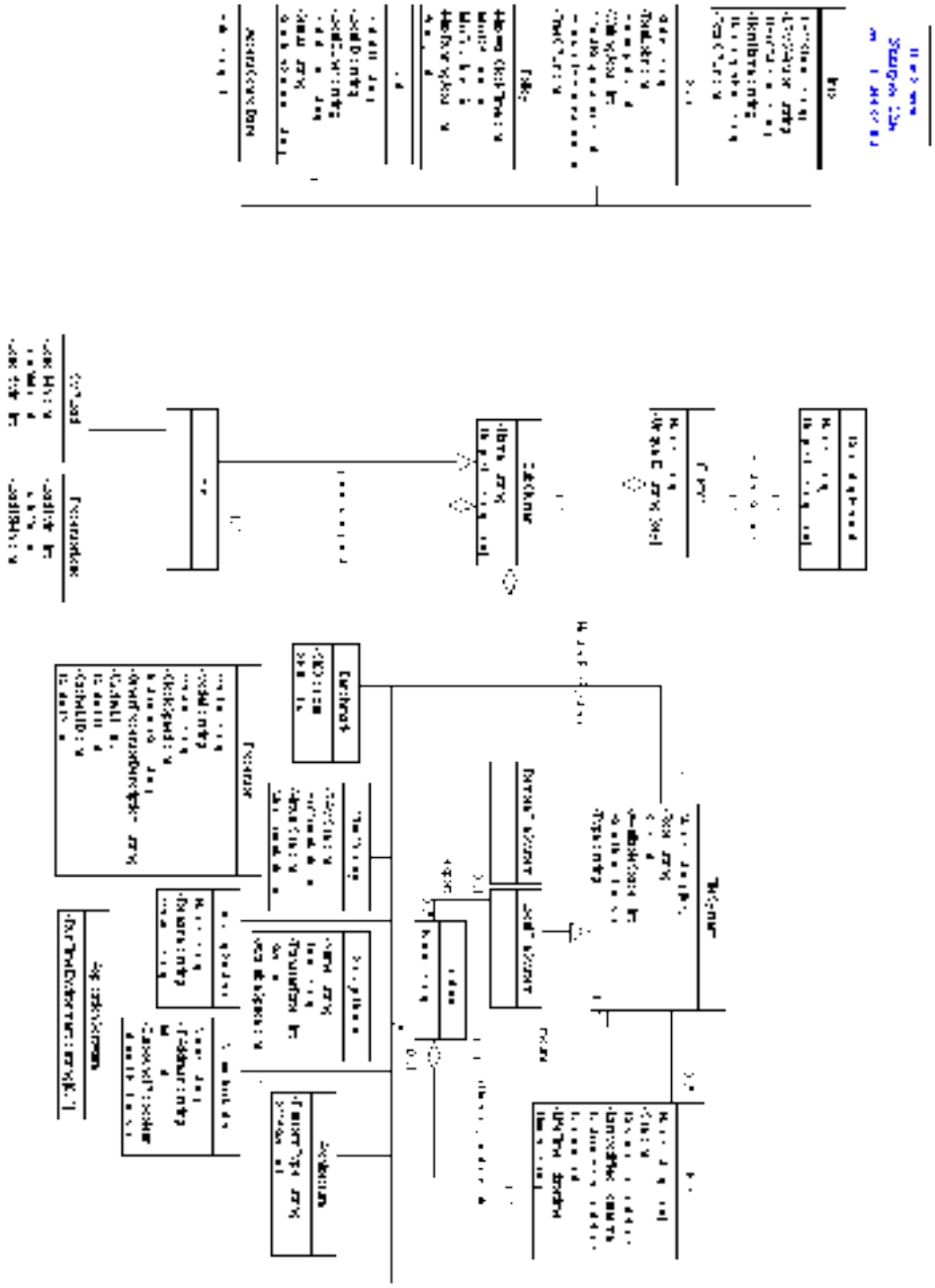

Figure 8 *– The GLUE Schema Of The Computing Element Of The Indexing Service*

(Figure borrowed from: http://www.cnaf.infn.it/~sergio/datatag/glue/CE/)

The information services also provide a registry of the grid services, which are available for client to use. A registry allows for soft-state registration of services. The services may register and update information as needed.

2.3.5 Grids And Peer-To-Peer Networks

The basic motivation behind the grid computing is sharing of some distributed resources between the participants over an overlay network. This is also the goal of peer-to-peer computing. As a matter of fact, some P2P networks have often been referred to as grids. Although, the grids and the P2P are solving the same issue of the distributed resource sharing there exist differences between these two technologies. In recent years, scientists are starting to recognize similarities and differences between these two technologies, in order to combine the positive aspects of both technologies [18], [19].

Development of grid computing has been driven by large physical organizations, such as universities and companies, which VOs consist of. The participants can be trusted and are well behaved. These large organization can afford expensive equipment and very good network connectivity. This is why diversity of shared resources within the grids is very high. Since high quality of service is required, special effort is made to maintain it. Deployment of new resources requires planning of resources, network infrastructures, etc, to achieve high availability and performance of the system. The deployments require lot of effort and are costly.

The grid systems are mostly used for intensive computations and data manipulations. This will result in high activities. These high activities, authentication requirements and sharing policies are some of reasons why the most topologies of the grid systems are centralized. This makes most grid systems unscalable.

On the other hand, we have the P2P networks which development has mostly been driven by the file-sharing communities. In these communities anonymity is highly valued and there are no trust assumptions. There are no requirements for a centralized administrative infrastructure. This makes the P2P systems highly scalable, with millions of participants. The cost of deployment of new resources in a P2P system is very low. No planning of infrastructures is necessary and deployments consist mostly of installation of new software.

Since most participants of the currently implemented P2P systems are private persons who can't afford an expensive equipment, the variety of the resources is very low. The file-sharing and sharing of computational power are the most popular services. The network connectivity is also poor. There are no quality of service guarantees in the P2P systems. All this contributes to low availability and performance of the system.

The vision described at the beginning of the section 2.3, requires properties of both technologies. It requires the diversity and the availability of resources, the high quality of service and the good performance of the system, which all are properties of a grid system. But it also requires the scalability, the low cost of deployment of new resources and the easy, "out-of-box" access to the system, which are properties of a P2P system.

3 Design

In large, global grid environments, the centralized meta-scheduling services might not be able to handle the large number of the GRAM instances and requests. There should exist some more scalable solution. A peer-to-peer-based meta-scheduling service would probably be more scalable and provide a better service.

A peer-to-peer-based meta-scheduling service is to be designed as a case study in this project. The service must be grid enabled, that is, all requests, responses, and notifications must follow the requirements of the OGSA and OGSI models. It should be able to act as a part of a larger grid system. The service must internally be based on a peer-to-peer system with the DHT functionality, particularly on the DKS system. A consequence of this requirement is that the system must be decentralized. There may not exist any centralized components and no single point of failure. The service must be scalable. In other words, the service must be able to handle, in a scalable manner, an increasing number of participating nodes. The algorithm for look-up of a lightly loaded node must be scalable.

3.1 The Model

The GRAM components are developed to manage job executions on a local system (a physical machine or a cluster of computers). A meta-scheduler is responsible for scheduling jobs on different installations of GRAM. It has the global view of the state of the distributed system and can, according to some algorithm, assign a job on the most suitable system. A meta-scheduler is not concerned with what jobs are and how they are executed; it is only concerned with how to distribute jobs so that original requirements on the meta-scheduler are achieved.

In this case, requirements for the meta-scheduler are that it should achieve an approximately well-balanced system. Also, this meta-scheduler must be able to handle one more parameter other then load. It should be able to assign jobs based on architecture required for the execution of a job.

Real systems are dynamic. Capacities of nodes are changing at any given time. A node might initiate a local execution, which will change the capacity of that node.

Also, jobs that are scheduled for execution in the GRAM might change capacity requirements at some point. This introduces a requirement for a dynamic load-balancing within the meta-scheduling system. In other words, a complete meta-scheduling service should be able to handle job migration between nodes, due to an overload. A design of such a service requires a larger effort. The time limitations of this project, however, demand a simplified model of the system.

In the simplified model, the capacities of nodes are constant and may not change in time. The jobs that arrive to the meta-scheduling service must have some maximum execution time. This time will never be exceeded and will always be constant, i.e. will never vary in time. With these assumptions, the system will never have a state such that a migration of a job is necessary. The meta-scheduling service should only consider the assignment of jobs to nodes. When a job is assigned to a certain node, it will execute and finish execution there.

If the system, at the moment of a request arrival, can't handle the execution of the requested job, i.e. all nodes are highly loaded and can not accept another job execution, it should consider the execution of that job as failed and notify the client that the execution of the job has failed because of the system saturation. The queuing of jobs should not be done. If necessary this functionality can be added to the system, but is not considered here because of the time restrictions.

In the conclusion, the service is a higher-order service, that will interconnect many instances of the GRAM in a scalable network. The primary goal is to achieve an approximately well-balanced system, where all nodes are almost equally loaded. From the client's point of view, it will help a client to find a GRAM instance which is most suitable for a job execution.

3.2 The Design

A very valuable property of a system is that it should be deployable in an already existing system with very little, if any, programming effort. As little changes to the existing system is made, the better. This produces an idea of making this meta-scheduling service transparent to the system. It is placed between the clients and the GRAM instances and none of them should really see any difference. Since the meta-scheduling service will hide a number of GRAM instances from clients, it will be very difficult to achieve the transparency for instance-specific

operations, such as life-time management and notification operations. It will require an larger effort, which is not allowed by time-restrictions. This is why we will concentrate on the transparency of the most important operation, the operation for instantiating a MJS, the *createService* operation.

The system must be able handle the dynamic nature of a P2P system. The participants of the service should be able to enter and leave as they wish without affecting the service in any crucial way. This introduces a requirement of many entry points. In particular, all nodes that participate in the P2P overlay network should also have the ability to receive a request for a job execution. If any of the nodes in the system fail, the service will not completely fail, as any other node in the system is able to replace it.

3.2.1 Client View

From a client point of view, this service should behave as any installation of GRAM. A client that is written to access the GRAM services, should be able to access this service with very little, if any, additional coding. The service should be transparent and all necessary information, which is needed for the scheduling decisions, should be extracted from the existing RSL description of a job.

When a client wishes to execute a job, it should first locate an entry point for the service by searching the local indexing service. When the scheduling service is located and a WSDL description is retrieved, a request may be sent to the entry point. The request is a createService operation, equivalent to the operation on the MJFS. The client will receive a Grid Service Handle (GSH) or a Grid Service Reference (GSR) as a result of the request. This will be a handle, or reference, of the MJS service that will be responsible for the execution of the job (see Figure 9). All further communication between the client and the MJS service will be performed directly.

If the requested resource is not available the client will be notified. In this case the resource is computational power. If the system is saturated, then the resource is not available. The system is saturated when there exists no node which can accept the job execution at the time of the request arrival, i.e. all nodes are highly loaded.

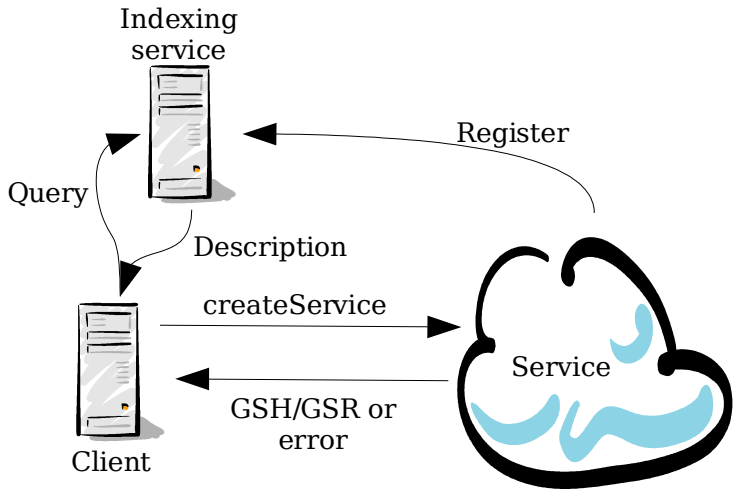

Figure 9 *– Client-side View Of The Service*

3.2.2 Service Architecture

Components

The service can be divided into two components: the Grid service and the P2P component (see Figure 10). The two components cover different functionalities of the service.

The Grid service component contains the code which defines the service. This is where the operation of the service are implemented. These operations should use the functionality provided by the P2P component. This component will interact with the Grid environment and thereby the clients.

When a request is received by the service, a search for a suitable node has to be performed and the request forwarded to that node. This is handled within the P2P component. This component uses the P2P overlay network of nodes (the DKS system) to execute an algorithm for a look-up of lightly loaded nodes and the code for forwarding of requests.

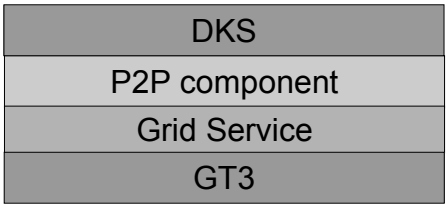

Figure 10 *– Components Of The Meta-scheduling Service*

All nodes participating in this meta-scheduling system have the same components. The nodes are all entry points for the service and can receive requests. The service does not depend on a particular set of nodes, but can exist no matter which nodes and how many of them are participating in the network. Clients can always access the service, even if most of the nodes fail.

Architecture Overview

When a request for job execution is received by an entry point, the service will first check whether the node that received the request has enough resources to execute the job. If so, the request is forwarded to the local installation of the GRAM. The service should try to minimize the amount of network traffic. It should strive to minimize the time-overhead when allocating jobs. If the node that received the request is highly loaded and can't accept another job execution, then a search for a node that can accept jobs must be performed.

The DKS overlay network is searched for a node that is lightly loaded. Four search schemes are presented in the section 3.2.4. All algorithms must have same interface, so that algorithms could be replaced without extra coding effort. The search is performed according to some scheme. If a lightly loaded node is found, the request will be forwarded, from within the search scheme, through peer-to-peer overlay network from the entry-point node to the lightly loaded node. If it is not found, the client should be notified about the saturation of the system. For more detailed discussion of searching schemes, see section 3.2.4.

The lightly loaded node will then, in its turn, forward the request to the local instance of the GRAM. The forwarding of the request through the P2P network is not necessary. The request can be sent directly to the GRAM instance on that node. By forwarding the request through the P2P overlay network, processing of requests can be, in some future work, easier added to the system. Processing of requests can be for example statistical processing (counting arriving requests), queuing of requests, etc.

When request arrives to the local GRAM instance it will be processed by the MMJFS. The result of the request, the GSH or the GSR of the newly created MJS service, will be returned to the node. This result must be sent back to the requesting client. The client expects to receive the result from the entry-point to which it has sent the request. To preserve the transparency of the service, the result must

be sent back to the node which served as an entry-point. The result will be forwarded through the P2P overlay network. This is done for the same reasons as for the request.

When the resulting GSH/GSR is received by the originated node, it will forward the GSH/GSR back to the client. From this point on, the client can communicate directly with the MJS service. The transparency is preserved and the client thinks at any time that it actually communicates with a GRAM instance. The described communication is depicted in Figure 11.

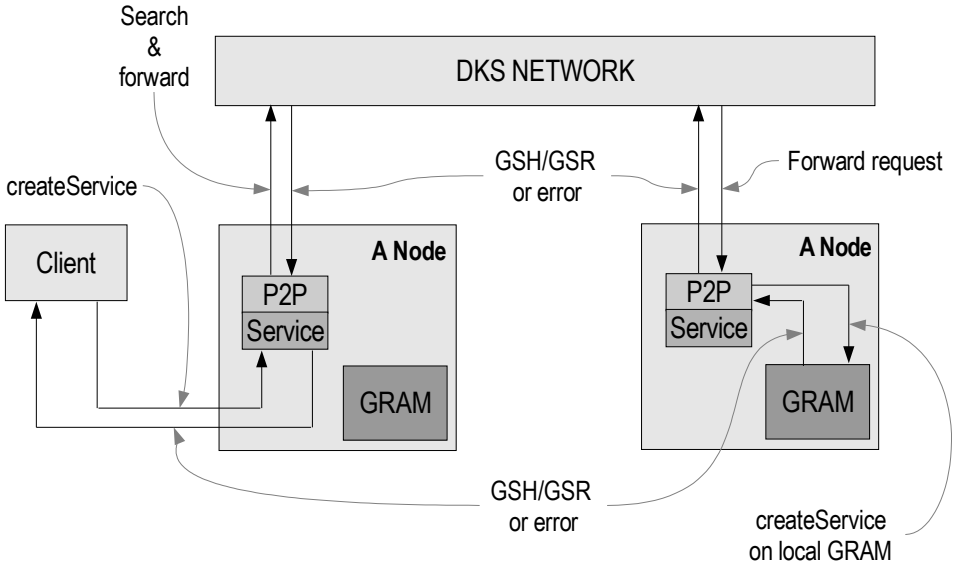

Figure 11 *– Job Allocation Scheme*

The Growth Of The P2P Overlay Network

The service is composed of many nodes which are connected and can communicate through a peer-to-peer overlay network. In any P2P system, nodes may arrive (i.e. join) and leave. The growth of the network must be handled in some way. Since the service will be a part of a grid environment, it's natural to design these administrative tasks as a service which has two operations: join and leave. The main advantage of the service-oriented approach is that the infrastructure, which is available in the Globus toolkit, for authentication and security can be used to authenticate nodes that wish to join the service.

The join operation in the DKS system is executed by the joining node. The service-oriented approach requires that the operation of joining is executed by a node that is already participating in the P2P network, on the joining node's behalf.

This is not supported by the current implementation of the DKS system, why designing the service-oriented approach would be somewhat complicated and would require a big effort. The time restrictions within the project doesn't allow that effort.

One solution is to let those operations be as is, and let any node that wishes to participate, to join without any authentication. This approach introduces the assumptions of the well-controlled access of the network and the well-behaved environment and community. These assumptions allow the security issues not to be considered.

3.2.3 Service Definition

To achieve the required transparency of the service, the service should be defined by the service definition of the MJFS in the GRAM. The WSDL description of the MJFS service and the corresponding Java interface are listed in Appendix A and only described in this section. The meta-scheduling service should implement all operations. This will make service easily deployed in new environments with very little, if any, programming effort.

MJFS is an OGSI compliant service. That means that all operations which are defined in the OGSI specification [13] are implemented by the service. These operations are defined according to the specification. The MJFS service is a factory service which can serve as notification sink and source. Therefore, it should implement the grid service operations, the factory service operations and the operations for notification sink and source. This is eight operations in total.

There are five operations defined for any OGSI compliant grid service. These operations are:

- *destroy* – Destroys the service
- *requestTerminationBefore* – requests change of the termination time of the service. The request specifies the latest desired termination time.
- *requestTerminationAfter* – requests change of the termination time of the service. The request specifies the earliest desired termination time.
- *setServiceData* – modifies a service data element's values.
- *findServiceData* – Queries the service data.

These operations perform the service data and lifetime management and should be applied on the service itself. When called, these operations should perform the service data and lifetime management on the meta-scheduling service. Since the meta-scheduling service does not keep any data elements, implementation of the service data management operations is not required. Since the meta-scheduling service should be a persistent service, the destruction of the service should not be handled by a client. So even lifetime management operations should not be implemented in meta-scheduling service. The definitions must exist, but the operations shouldn't do anything.

There are two notification source and sink operations. These are:

- *subscribe* – subscribes the service for notifications from another service.
- *deliverNotification* – registers a client as a receiver of notifications.

The operation *subscribe* must be defined in any service that wishes to send out notifications. The operation *deliverNotification* must be defined in any service that wishes to receive notifications from another service. The meta-scheduling service should not send out any notifications, so the notification source operation shouldn't be implemented. The service could subscribe for some notifications from another service, such as notifications from indexing service. How the service will retrieve information from indexing service is a question of an implementation.

The most important operation of the eight operations which are defined in MJFS, is the factory service operation, *createService* operation. This is the operation which is called when a client wishes to initiate an execution of a job. It needs to request from the MJFS an instance of the MJS service. The operation takes one parameter, the RSL specification of the job, which is to be executed. This operation must be implemented in the meta-scheduling service. When the operation is invoked by a client, the meta-scheduling service should perform the allocation of the job.

All operations are defined by the OGSI specification. For more detailed description of the operations, see [13]. For further information on the functionality of the MJFS service please see [29].

3.2.4 Searching For A Node

When a request is received by a node, which serves as entry-point, it will first check if the job can be executed locally. If not, another suitable node must be found somewhere in the system. The search is performed according to some scheme. Four schemes are described below. Three of them are based on the schemes presented in [31]. Two of those four schemes will be described, but not considered, since they are out of the scope of this project. The interface of these schemes must be unified. The replacement of a scheme should not require large programming effort.

In any scheme, the system must decide whether a node is suitable for execution. The decisions are based on several parameters. First, a node should not accept a job execution if its load is high. Only nodes with a low level of load should accept a job execution. Second, a node should not accept a job execution if its resources don't fit the requirements specified in the RSL specification of the job. These requirements can include any parameter available in the definition of the RSL language.

Extracting Parameters

When a client calls the *createService* operation it has to supply, a RSL specification of resources that execution of the job requires. This specification contains the information that can be extracted and used for making the scheduling decisions. The execution may depend on several different parameters, not only on load. A job, for example, might demand a certain architecture of the node where the execution will be performed.

The parameters, which a job requires, can be extracted from the RSL specification by parsing the XML code. If a parameter is needed for the scheduling decision, but is not provided by the client in the RSL specification, the default values should be specified. For example, the default value for an instruction set of a processor might be i386.

Making A Decision

The meta-scheduling service must primarily take into account the load of nodes. This is crucial for providing the approximately load-balanced system. In other hand, the meta-scheduling service may take into account other parameters specified in the RSL specification.

A load of a node (processor) can be retrieved from the indexing services. A load of a processor is specified by the three averaged values, during one, five and 15 minutes. Which value the service will use, depends on how responsive the system should be, and is a matter of an implementation. In the further discussions, we will assume that a current load of a processor is

$$l = ce{:}Last15Min.$$

The wanted level of the load, L, should be specified on every node. This level is the utilization which is wanted on that node. The lightly and heavily loaded nodes are then defined as following:

$$\text{The heavily loaded node: } l \geq L$$

$$\text{The lightly loaded node: } l < L$$

A node should only accept a job execution if and only if that node is lightly loaded.

Other parameters that will be used for making a scheduling decision may be used to provide more accurate assigning of jobs. Parameters that are considered must be available both in the RSL definition and in the information services. The RSL can be extended if necessary, so the limitations are in the information services.

One-to-one Scheme

This scheme is based on a direct communication between nodes. The P2P network is used only for routing of messages and connecting the nodes. The network is searched randomly for a lightly loaded node. To locate a lightly loaded node a random identifier is chosen and the node responsible for that identifier is contacted. A request for job execution is sent. The decision whether to accept or decline the request is made by the contacted node, locally. If it accepts, it should forward the request to the local GRAM instance. If the contacted node didn't accept the job execution, the next random node is chosen and contacted (see Figure 12). The process repeats until a node accepts a job or the whole system is searched.

This scheme has the advantage of the simplicity of the algorithm. There are no repositories that must be maintained and there is no advertisement of the nodes properties. The decisions are made locally and are based on the current state of the node, rather then on a state which was advertised some time ago. There are no racing conditions that must be considered, as well. In most environments random selections of nodes will produce a very good hit-rate.

The downside of this scheme is that any node might be contacted, not only highly loaded nodes, but even those that don't match other requested properties, such as the architecture. These nodes are not suitable no matter what their load is. This will produce a higher network traffic.

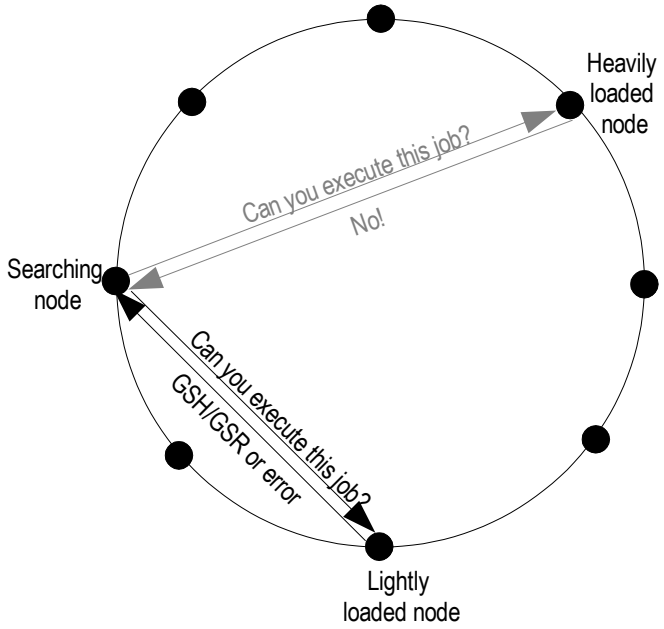

Figure 12 *– The One-to-one Scheme*

One-to-many Scheme

This scheme is based on maintaining repositories, the directories, of lightly loaded nodes. The directories should contain the information about the load of nodes as well as other parameters which should be used when making the scheduling decisions. Such parameters could be the architectures of nodes, etc. There should exist *d* directories, where *d* is much less then the number of nodes in the system, *N*. A well-known hash function *h'* is used to hash a lightly loaded node into a directory. The hash function *h'* should result in the interval [0, d[. A directory *j* is hashed by using another well-known hash function *h*, and then stored on the node responsible for the identifier given by *h(j)*. A lightly loaded node *l* advertise its load and other parameters to a node responsible for the identifier

$$i = h(h'(l)).$$

When a node shifts from a lightly loaded state to a highly loaded state it should remove itself from the directory. This should be done by advertising a high load to the node responsible for the directory. The node should then remove the advertising node from the directory. The advertisement of the nodes is depicted in Figure 13.

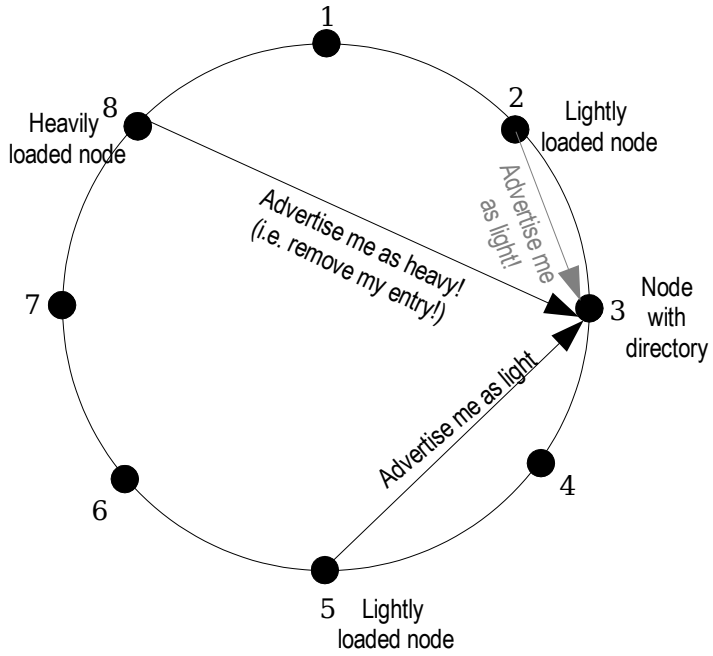

Figure 13 *– The One-to-many Scheme: Node Advertisement*

Nodes in a P2P system might fail, i.e. leave without cleaning up. This is why an entry in a directory should be time-limited. If a lightly loaded node didn't advertise itself within a time limit T, its entry should be removed. To be sure that they are listed in a directory, nodes should advertise themselves periodically, with a period T/2. This will, eventually, remove all unavailable nodes from the directories.

When the system is searched for a lightly loaded node, the searching node picks a random directory *j*. The query is sent to the node which is responsible for that directory, i.e. is responsible for the identifier *h(j)*. The directory is searched for nodes that match the requirements specified in the query. A list of possible nodes is sent back to the searching node (see Figure 14).

The information that was retrieved from the directory might be not updated. The nodes might actually be highly loaded. In the worst case, the node could have failed and left garbage in the directory. A request for a job execution is sent to the node. If the node is still participating, it will make a decision whether it wishes and is able to accept the job execution. If it accepts, it should forward the request to the local GRAM instance. If it rejects the request, the searching node should contact the next lightly loaded node in the list, and so on. If the list does not contain any lightly loaded nodes, the next random selected directory is queried. The

process continues until the whole system is searched without result (see Figure 14).

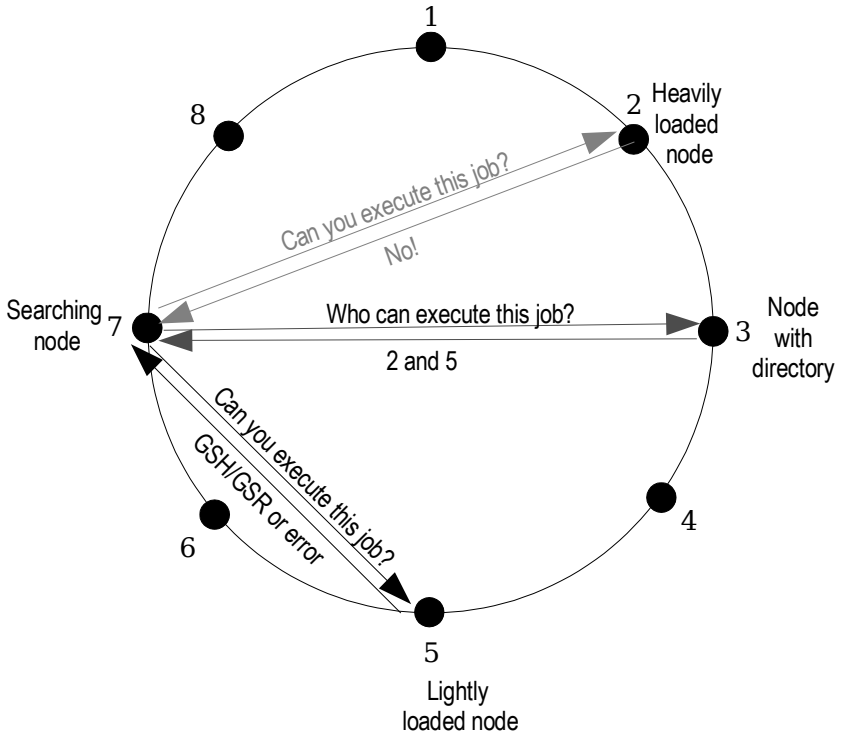

Figure 14 *– The One-to-many Scheme: Searching For A Lightly Loaded Node*

The DHT functionality of the P2P network is used in this scheme to store the directories. The directories provide filtering, so that the nodes we're not interested in are not bothered with the requests. The downside is that very a little amount of nodes is handling the directories and will have to communicate intensively with all other nodes. In a very busy environment this might cause an overload due to processing of the large amount of queries. A solution is to increase the number of directories, so that the load due to processing queries is spread out. But this number may not become too large, because when the number of directories is comparable to the number of nodes, this scheme will reduce to an inefficient variant of the one-to-one scheme.

The Many-to-many Scheme

The many-to-many scheme is further development of the previous scheme, the one-to-many scheme. In this scheme, besides the directories of lightly loaded nodes, directories of jobs waiting for executions are also kept in the DHT. If a node is lightly loaded it should search the directories of pending jobs to find a job

it can execute. Both the entry-point nodes and the lightly loaded nodes perform the search (see Figure 15). This should reduce the search time. As a consequence of the pending jobs directories, the system will keep on accepting the requests and advertise the jobs, even if the system is saturated.

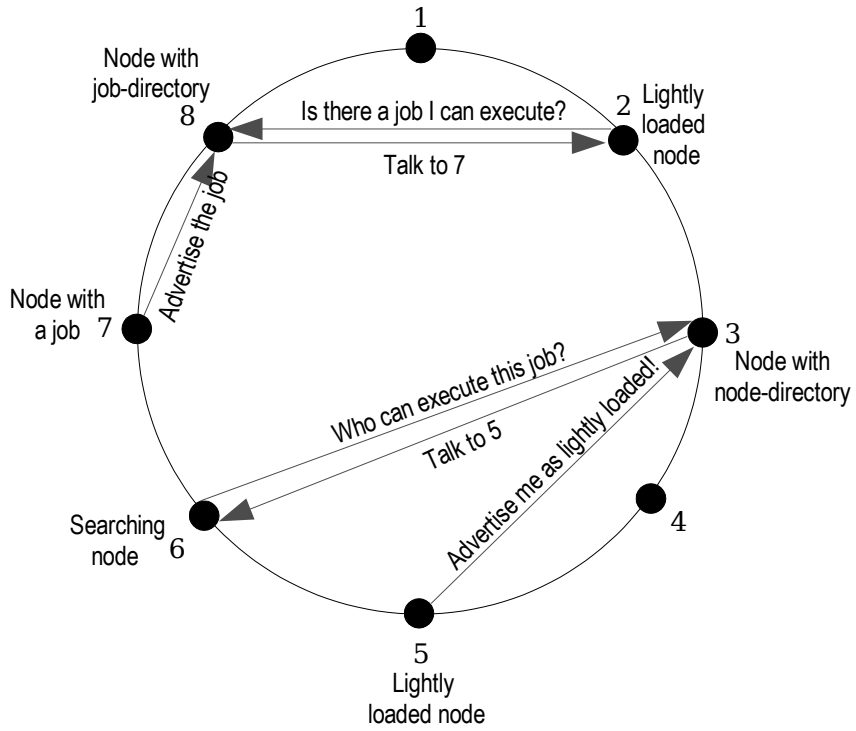

Figure 15 *– The Many-to-many Scheme*

The advertisement of the pending jobs requires that the requests for job executions should be queued somewhere, until nodes, on which they will be executed, are found. This queuing is the actual reason the system will keep on accepting the requests although saturated. But, requiring the introduction of the queues in the system is against the assumption that was made at the beginning of this chapter. This is why this scheme is out of the scope of this project and will not be considered further.

The Broadcast Scheme

In this scheme querying of the system for lightly loaded nodes is performed by broadcasting a message into the system. The k-ary spanning tree in the DKS could be used to filter out the heavily loaded nodes. The spanning tree, in the DKS would allow us to perform the filtering of the querying results in a scalable manner and the node, from which the request originates, would receive a small number of messages compared to the number of participating nodes (see Figure 16).

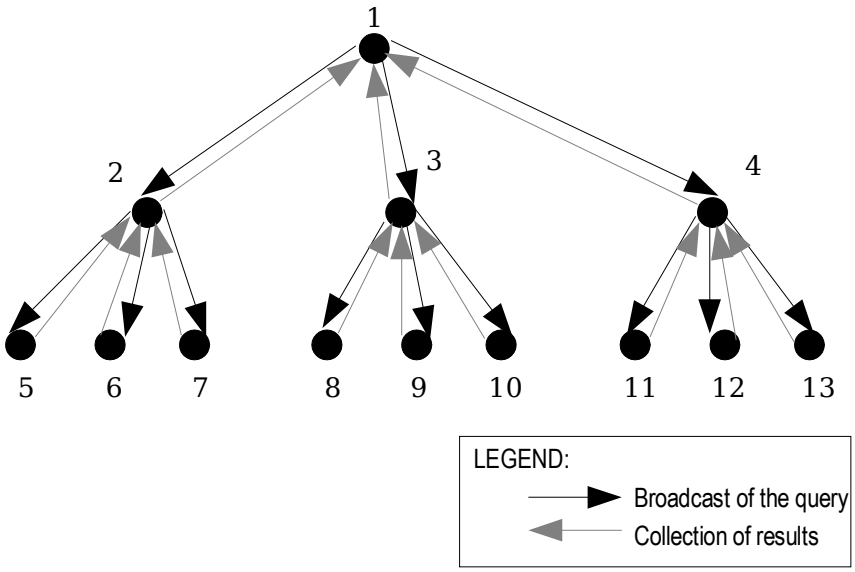

Figure 16 *– The Broadcast Scheme*

The current DKS implementation doesn't provide an access to the K-nary spanning tree of the P2P network. This means that the broadcasting scheme can't be realized in the way described above. The only way a broadcast could be used is to broadcast a request and let the whole system answer back to the requester, basically making a snapshot of the system. But this is not desirable, as thousands and thousands of messages could flood the requester and thereby creating a DDOS attack unintentionally. This version of the algorithm could be classified as dangerous for the participating nodes.

The broadcasting scheme will not be considered further in this project.

4 Implementation Issues

The implementation of the meta-scheduling service should be done in the Java programming language. The main components that should be used for realization of the service are the Globus Toolkit 3.2 and the DKS system. The current implementation of the DKS is developed in the Java, which somewhat dictates the requirement of what development tools should be used when implementing the service.

4.1 Classes

There should exist two classes that cover the functionality of the meta-scheduling service. The functionality should be divided according to the components described in section 3.2.2. The classes presented in the following subsections are the basic classes in an implementation of the service. There will exist other classes that are necessary for use of the DKS system, such as a class that extends the *DKSMessageReceiver* class. These classes could also be realized as subclasses.

4.1.1 Grid Service Component

The first class extends the service implementation, *GridServiceImpl*, and implements the interface of the MJFS, *MasterManagedJobFactoryService*. This class should implement only the functionality of a grid service. No reference to peer-to-peer network should be included. This class should only use the functionality provided by the P2P component. The outline of the class can be found below.

Outline Of The MetaScheduler Class

```
Please observe that this is an outline and not a complete and
correct code.

public class MetaScheduler extends GridServiceImpl implements
MasterManagedJobFactoryService {
  public MetaScheduler() {};
  public void createService(...) {}
  public void deliverNotification(...) {}
  public void subscribe(...) {}
  public ExtensibilityType setServiceData(...) {}
  public void destroy() {}
```

```
  public TerminationTimeType requestTerminationAfter(...) {}
  public TerminationTimeType requestTerminationBefore(...) {}
  public ExtensibilityType findServiceData(...) {}
}
```

Some of the methods might be implemented as empty calls, that is, the operations do nothing. The method *createService* must be implemented in any implementation because this method performs the operation of scheduling.

4.1.2 Peer-to-peer Component

The second class should implement the functionality related to the peer-to-peer overlay network. This includes the joining, the leaving, the searching schemes and the forwarding of requests, etc. A proposition of a possible outline of the class is provided below.

Outline Of The P2PComponent Class

```
Please observe that this is an outline and not the complete and
correct code.

public class P2PComponent {
  public P2PComponent () {};
  public forwardRequest(...) {}
  public execute(...) {}
  public boolean isExecutable(...) {}
}
```

In the class above the basic methods are specified. The method *forwardRequest* is used for searching for an appropriate node and the forwarding of a request to it. It should return a result of running the *createService* operation on a local instance of the GRAM on some node. If no available nodes are found, it should raise an exception. The method *execute* should execute the *createService* operation on the local instance of the GRAM for the specified job. The method *isExecutable* should check if the specified job can be executed on the node.

The class should be implemented for each searching scheme separately, although the peer-to-peer component contains some functionality which is independent of searching scheme. It is necessary to mix these functionalities to reduce the number of messages which are sent between the nodes.

4.2 Joining And Leaving

At the creation time of the service, the code for joining the newly created node must be executed. The constructor of the P2PComponent class must join the node. If the node is to be joined into the overlay network, it must know about at least one node that already participates in it. This information must either be hard-coded, provided as a parameter when starting the service or saved in a configuration file. A private method may be added to the P2PComponent class to handle the joining. This method could be used for other start-up tasks. At start-up, the node might wish to advertise itself into a directory, so that it can receive requests from the very beginning. This is not necessary, though, since the node will frequently check its state and advertise itself if necessary.

A node should call the leave operation in the DKS system when the service is about to be shut down. This is not required, though. A node should be able to leave the network without calling the leave operation and the system must handle it. The system must be able to handle failing nodes. A private method may be added to the P2PComponent class to handle the leaving. This method could perform other tasks associated with the shutdown of the service, such as advertising a high load. An advertisement of a high load will effectively remove a node from any directory.

4.3 P2P Messaging

When searching the P2P overlay network for lightly loaded nodes some messaging will have to be performed. In the two schemes described in previous section, three messages are used:

- Request job execution
- Advertise
- Query

The messages could be XML-based, since all other technologies are using the XML-based messaging protocols. A parser for XML code must be included in an implementation, since the RSL specifications must be parsed. The use of the XML in the P2P messaging will not demand much more programming effort.

A request for a job execution should contain a RSL specification of the job, so that a node can extract the relevant data and decide whether it can execute the job. Besides the RSL specification no other data has to be sent. So, the RSL spec-

ification can serve as a request message. To be sure that there are no misunderstandings of the meaning of the message, the RSL specification can be wrapped into a '<request>' tag, as example shows.

Request Message

```
<request>
  <rsl:rsl>
    ...
  </rsl:rsl>
</request>
```

When a request is received by a node, it should check whether it can provide the resources specified in the RSL specification. If it can, then it should contact the local GRAM instance and request creation of MJS and return the resulting GSH/GSR. If it can't provide the requested resources, it should return a reject message. Outlines of these messages are proposed below.

Request Result Message

```
Result message can have two different outlines. First, the
result of creation of MJS service:
  <request:result>
    <reference>
      ...
    </reference>
  </request:result>

Second, when the request is rejected by the node:
  <request:result>
    <reject/>
  </request:result>
```

The querying message should contain the RSL specification, as well, since the filtering of a directory will be performed according to it. Beyond this, no other data is necessary for the querying. Also this message must be wrapped so that no misunderstanding can arise.

Querying Message

```
<query>
  <rsl:rsl>
    ...
  </rsl:rsl>
</query>
```

When a node receives a querying message, it should search the directory and create a list of all nodes which meet the criteria specified in the RSL specification. The list should be returned as a resulting message. A possible outline of the resulting message is shown below.

Query Result

```
<query:result>
  <node id="..."/>
  <node id="..."/>
  <node id="..."/>
  ...
</query:result>
```

The advertising message have several purposes. The message has the purpose of advertising a lightly loaded node. The second purpose is to keep an entry updated. At the same time, it serves as a keep-alive message that will keep an entry in the directory. Also, this message can be used to remove a node from a directory by advertising a node as heavy. The message must include, at least, the following information:

- Identifier – The unique identifier in the P2P overlay network
- Load level – [High/low] Whether advertising high or low load
- Load – Current load of node
- Preferred load – Preferred load of node

More information can be added. This information could be other parameters of a node, such as an architecture, etc.

Advertisement Message

```
<advertise id="..."
            level=[high | low]
            load="..."
            preferred="..." />
```

When a node receives an advertisement message it should first check if advertisement is for a high or a low load. If it is for a low load, then it should add the advertising node into the directory it is responsible for. If the node is already in the directory, it should update, in the entry, the time-stamp, the load information and other information about a node. If the advertisement message is advertising a high level of load, the node responsible for the directory should remove the advertising node from the directory.

4.4 Searching

A searching scheme is implemented in the method *forwardRequest*. The method should perform the search and, at the same time, forward the request to the found, lightly-loaded node. The merging of these two apparently different tasks is necessary for the reduction of messaging in the overlay network. When directly forwarding a request, there is no need for reservation of resources or such. The request for a job execution can be rejected in two levels: first by a peer-to-peer node and, second, by the GRAM. The rejection from a peer-to-peer node should result in a further search. The message of this rejection is defined in section 4.3. The rejection from GRAM, should be forwarded back to the client.

In the *forwardRequest* method, a searching scheme should be implemented. Because of its simplicity, the one-to-one scheme can be implemented within this method. On the other hand the one-to-many scheme is more complex and requires the extra functionality. Additional, private, methods should be introduced to handle the directories. Also a data-structure must be introduced to handle the entries in the directories.

4.4.1 Directories

The directories can be implemented using any data structure: linked lists, hash tables, etc. A special class may be defined which will be used as a data-element in the directories. This data element must contain the following fields:

- Identifier – The unique identifier in the P2P overlay network
- Load – Current load of node
- Preferred load – Preferred load of node

These are basic parameters, which are used when deciding whether a job could be executed on this node, and must exist in a directory. Additional fields may be in-

troduced if other parameters, such as an instruction set, are used for the decision-making.

An advertisement message must result in an update of the directory. A node should be removed from the directory if it advertises a high load, and the entry should be updated if it advertises a low load. This can be handled by a single method, *updateDirectory,* since update of an entry is simply removing and adding that entry.

When a query message arrives, the directory must be searched for nodes that fit the query. This can be done in a separate method, *queryDirectory*. A list of nodes should result by calling this method. The list should only contain identifiers of nodes. No other information is necessary.

4.5 Listening

When receiving messages in the DKS system, a listener class is used for handling the event. A listener class must extend the *DKSMessageReceiver* class, which is defined by the DKS implementation. The listener class can be defined as subclass or in-line class. The possible messages that could be received by a node are defined in section 4.3.

When a node receives a message the listener class handles it by calling appropriate functions. If the request-for-execution message is received, then the node should call the method *isExecutable*, to check if the job can be executed on the node. If it can, then the *execute* method should be called and the result forwarded back to the requesting node. If the advertisement message is received, the *updateDirectory* method should be called, to update the entry in the directory, which this node is responsible for. If the querying message is received, then the *queryDirectory* method should be called to build up a list of nodes.

5 Conclusions

In this project, three technologies have been studied: the grids, the web services and the peer-to-peer overlay networks. In the second phase of the project, we have tried and succeeded in designing a grid service, which is based on the peer-to-peer overlay network technology, and provided the architecture. This has resulted in a grid service that is scalable, self-organized, fault-tolerant and distributed. The service can handle failing nodes and it has no single point of failure. The service is a meta-scheduling service for the GRAM. Two resource-searching schemes have been considered: the one-to-one and the one-to-many scheme.

From the work presented in the report, we can conclude that the peer-to-peer overlay networks with the Distributed Hash Tables (DHTs) functionality, in particular the DKS, can be used within a grid environment for a realization of grid services.

5.1 Future Work

The implementation of the design is a natural step in the work. The implementation will provide a working prototype, which can then be tested and evaluated. The results of the future work must show whether the design has the desirable performances.

A long term goal is to provide a more complex design and implementation of the meta-scheduling service. Also, the goal is to incorporate the peer-to-peer overlay network technology in other services, which are today available in a grid environment.

6 Appendix A

6.1 Service Definitions

The meta-scheduling service must have the same interface as the MMJFS. The port-type definition should be basically the same. The port-type definition of the MMJFS is listed below. The associated Java interface is listed, as well.

Port-type Definition Of The MMJFS In The GRAM

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MasterManagedJobFactoryService"
             targetNamespace="http://www.globus.org/namespaces/2003/04/mmjfs"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:gwsdl="http://www.gridforum.org/namespaces/2003/03/gridWSDLE
xtensions"
             xmlns:ogsi="http://www.gridforum.org/namespaces/2003/03/OGSI"
             xmlns:tns="http://www.globus.org/namespaces/2003/04/mmjfs"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<import location="../../ogsi/ogsi.gwsdl"
        namespace="http://www.gridforum.org/namespaces/2003/03/OGSI"/>

<portType name="MasterManagedJobFactoryService">
  <operation name="createService">
    <input message="ns36:CreateServiceInputMessage"
           xmlns:ns36="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    <output message="ns37:CreateServiceOutputMessage"
            xmlns:ns37="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    <fault message="ns38:ExtensibilityNotSupportedFaultMessage"
           name="ExtensibilityNotSupportedFault"
           xmlns:ns38="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    <fault message="ns39:ExtensibilityTypeFaultMessage"
           name="ExtensibilityTypeFault"
           xmlns:ns39="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    <fault message="ns40:ServiceAlreadyExistsFaultMessage"
           name="ServiceAlreadyExistsFault"
           xmlns:ns40="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    <fault message="ns41:FaultMessage"
           name="Fault"
           xmlns:ns41="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  </operation>
  <operation name="deliverNotification">
    <input message="ns35:DeliverNotificationInputMessage"
```

```
            xmlns:ns35="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
        </operation>
        <operation name="subscribe">
          <input message="ns0:SubscribeInputMessage"
                 xmlns:ns0="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <output message="ns1:SubscribeOutputMessage"
                  xmlns:ns1="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns2:ExtensibilityNotSupportedFaultMessage"
                 name="ExtensibilityNotSupportedFault"
                 xmlns:ns2="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns3:ExtensibilityTypeFaultMessage"
                 name="ExtensibilityTypeFault"
                 xmlns:ns3="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns4:TargetInvalidFaultMessage"
                 name="TargetInvalidFault"
                 xmlns:ns4="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns5:FaultMessage"
                 name="Fault"
                 xmlns:ns5="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
        </operation>
        <operation name="setServiceData">
          <input message="ns48:SetServiceDataInputMessage"
                 xmlns:ns48="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <output message="ns49:SetServiceDataOutputMessage"
                  xmlns:ns49="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns50:ExtensibilityNotSupportedFaultMessage"
                 name="ExtensibilityNotSupportedFault"
                 xmlns:ns50="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns51:ExtensibilityTypeFaultMessage"
                 name="ExtensibilityTypeFault"
                 xmlns:ns51="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns52:CardinalityViolationFaultMessage"
                 name="CadinalityViolationFault"
                 xmlns:ns52="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns53:MutabilityViolationFaultMessage"
                 name="MutabilityViolationFault"
                 xmlns:ns53="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns54:ModifiabilityViolationFaultMessage"
                 name="ModifabilityViolationFault"
                 xmlns:ns54="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns55:TypeViolationFaultMessage"
                 name="TypeViolationFault"
                 xmlns:ns55="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns56:IncorrectValueFaultMessage"
                 name="IncorrectValueFault"
                 xmlns:ns56="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
          <fault message="ns57:PartialFailureFaultMessage"
                 name="PartialFailureFault"
                 xmlns:ns57="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
```

```
  <fault message="ns58:FaultMessage"
         name="Fault"
         xmlns:ns58="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
</operation>
<operation name="destroy">
  <input message="ns67:DestroyInputMessage"
         xmlns:ns67="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <output message="ns68:DestroyOutputMessage"
          xmlns:ns68="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns69:ServiceNotDestroyedFaultMessage"
         name="ServiceNotDestroyedFault"
         xmlns:ns69="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns70:FaultMessage"
         name="Fault"
         xmlns:ns70="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
</operation>
<operation name="requestTerminationAfter">
  <input message="ns59:RequestTerminationAfterInputMessage"
         xmlns:ns59="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <output message="ns60:RequestTerminationAfterOutputMessage"
          xmlns:ns60="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns61:TerminationTimeUnchangedFaultMessage"
         name="TerminationTimeUnchangedFault"
         xmlns:ns61="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns62:FaultMessage"
         name="Fault"
         xmlns:ns62="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
</operation>
<operation name="requestTerminationBefore">
  <input message="ns63:RequestTerminationBeforeInputMessage"
         xmlns:ns63="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <output message="ns64:RequestTerminationBeforeOutputMessage"
          xmlns:ns64="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns65:TerminationTimeUnchangedFaultMessage"
         name="TerminationTimeUnchangedFault"
         xmlns:ns65="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns66:FaultMessage"
         name="Fault"
         xmlns:ns66="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
</operation>
<operation name="findServiceData">
  <input message="ns42:FindServiceDataInputMessage"
         xmlns:ns42="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <output message="ns43:FindServiceDataOutputMessage"
          xmlns:ns43="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns44:ExtensibilityNotSupportedFaultMessage"
         name="ExtensibilityNotSupportedFault"
         xmlns:ns44="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
  <fault message="ns45:ExtensibilityTypeFaultMessage"
```

```
            name="ExtensibilityTypeFault"
            xmlns:ns45="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
      <fault message="ns46:TargetInvalidFaultMessage"
            name="TargetInvalidFault"
            xmlns:ns46="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
      <fault message="ns47:FaultMessage"
            name="Fault"
            xmlns:ns47="http://www.gridforum.org/namespaces/2003/03/OGSI"/>
    </operation>
</portType>
<gwsdl:portType extends="ogsi:NotificationSource
ogsi:NotificationSink                              ogsi:Factory"
                  name="MasterManagedJobFactoryService"/>

</definitions>
```

Java Interface For MJFS In GRAM

```
/**
 * MasterManagedJobFactoryService.java
 *
 * This file was auto-generated from WSDL
 * by the Apache Axis WSDL2Java emitter.
 */

package org.globus.ogsa.base.gram.mmjfs;

public interface MasterManagedJobFactoryService extends
org.gridforum.ogsi.GridService {

    public void createService(org.gridforum.ogsi.TerminationTimeType
terminationTime, org.gridforum.ogsi.ExtensibilityType creationParameters,
org.gridforum.ogsi.holders.LocatorTypeHolder locator,
org.gridforum.ogsi.holders.TerminationTimeTypeHolder currentTerminationTime,
org.gridforum.ogsi.holders.ExtensibilityTypeHolder extensibilityOutput) throws
java.rmi.RemoteException, org.gridforum.ogsi.ServiceAlreadyExistsFaultType,
org.gridforum.ogsi.ExtensibilityTypeFaultType,
org.gridforum.ogsi.ExtensibilityNotSupportedFaultType,
org.gridforum.ogsi.FaultType;

    public void deliverNotification(org.gridforum.ogsi.ExtensibilityType
message) throws java.rmi.RemoteException;

    public void subscribe(org.gridforum.ogsi.ExtensibilityType
subscriptionExpression, org.gridforum.ogsi.LocatorType sink,
org.gridforum.ogsi.ExtendedDateTimeType expirationTime,
org.gridforum.ogsi.holders.LocatorTypeHolder subscriptionInstanceLocator,
org.gridforum.ogsi.holders.TerminationTimeTypeHolder currentTerminationTime)
throws java.rmi.RemoteException, org.gridforum.ogsi.TargetInvalidFaultType,
org.gridforum.ogsi.ExtensibilityTypeFaultType,
org.gridforum.ogsi.ExtensibilityNotSupportedFaultType,
org.gridforum.ogsi.FaultType;

    public org.gridforum.ogsi.ExtensibilityType setServiceData
(org.gridforum.ogsi.ExtensibilityType updateExpression) throws
java.rmi.RemoteException, org.gridforum.ogsi.ModifiabilityViolationFaultType,
org.gridforum.ogsi.PartialFailureFaultType,
org.gridforum.ogsi.TypeViolationFaultType,
org.gridforum.ogsi.MutabilityViolationFaultType,
org.gridforum.ogsi.CardinalityViolationFaultType,
org.gridforum.ogsi.ExtensibilityTypeFaultType,
org.gridforum.ogsi.IncorrectValueFaultType,
org.gridforum.ogsi.ExtensibilityNotSupportedFaultType,
org.gridforum.ogsi.FaultType;
```

```
    public void destroy() throws java.rmi.RemoteException,
org.gridforum.ogsi.ServiceNotDestroyedFaultType, org.gridforum.ogsi.FaultType;

    public org.gridforum.ogsi.TerminationTimeType requestTerminationAfter
(org.gridforum.ogsi.ExtendedDateTimeType terminationTime) throws
java.rmi.RemoteException, org.gridforum.ogsi.TerminationTimeUnchangedFaultType,
org.gridforum.ogsi.FaultType;

    public org.gridforum.ogsi.TerminationTimeType requestTerminationBefore
(org.gridforum.ogsi.ExtendedDateTimeType terminationTime) throws
java.rmi.RemoteException, org.gridforum.ogsi.TerminationTimeUnchangedFaultType,
org.gridforum.ogsi.FaultType;

    public org.gridforum.ogsi.ExtensibilityType findServiceData
(org.gridforum.ogsi.ExtensibilityType queryExpression) throws
java.rmi.RemoteException, org.gridforum.ogsi.TargetInvalidFaultType,
org.gridforum.ogsi.ExtensibilityTypeFaultType,
org.gridforum.ogsi.ExtensibilityNotSupportedFaultType,
org.gridforum.ogsi.FaultType;

}
```

7 List Of Abbreviations

CE – Computing Element
DDOS – Distributed Denial Of Service
DHT – Distributed Hash Table
DKS – Distributed K-nary System
FSFS – File Stream Factory Service
FSS – File Stream Service
GRAM – Globus Resource Allocation Manager
GSH – Grid Service Handle
GSI – Grid Security Infrastructure
GSR – Grid Service Reference
GT – Globus Toolkit
GT3 – Globus Toolkit, version 3
MDS – Management and Discovery Service
MHE – Master Hosting Environment
MJFS – Managed Job Factory Service
MJS – Managed Job Service
MMJFS – Master Managed Job Factory Service
OGSA – Open Grid Service Architecture
OGSI – Open Grid Service Interface
P2P – Peer-To-Peer
RSL – Resource Specification Language
SDE – Service Data Element
SOAP – Simple Object A Protocol
UDDI – Universal Description and Integration
UHE – User Hosting Environment
VO – Virtual Organization
WSDL – Web Services Description Language
XML – Extensible Markup Language

8 References

[1] *"Peer-To-Peer Computing"*, Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, Zhichen Xu, HP Labratories Palo Alto, HPL-2002-57, March 2002

[2] *"A Framework For The Understanding, Optimization and Design Of Structured Peer-To-Peer Systems"*, Sameh El-Ansary, Licentiate of Philosophy Dissertation at Royal Institute of Technology, 2003

[3] *"What Is P2P ... And What Isn't"*, Clay Shirky, O'Reilly OpenP2P.com, http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

[4] Web services, URL: http://www.w3.org/2002/ws/

[5] *"New to SOA and Web services"*, IBM developerWorks, http://www-106.ibm.com/developerworks/webservices/newto/websvc.html

[6] *"Web Services Architecture"*, W3C Working Group Note 11 February 2004, http://www.w3.org/TR/ws-arch/

[7] *"Web Services Description Language (WSDL) 1.1"*, Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[8] *"SOAP Version 1.2 Part 0: Primer"*, Nilo Mitra, W3C Recommendation 24 June 2003, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/

[9] *"Web Services Conceptual Architecture"*, Heather Kreger, IBM Software Group, May 2001, http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf

[10] *"Distributed Computing Research Issues in Grid Computing"*, Henri Casanova, in Quarterly Newsletter for the ACM Special Interest Group on Algorithms and Computation Theory (SIGACT News), Vol. 33, Num. 2, Sept. 2002.

[11] *"The Anatomy of the Grid: Enabling Scalable Virtual Organizations"*, I. Foster, C. Kesselman, S. Tuecke, International J. Supercomputer Applications, 15(3), 2001.

[12] *"The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration"*, I. Foster, C. Kesselman, J. Nick, S. Tuecke, OpenGrid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[13] "*Open Grid Services Infrastructure (OGSI) Version 1.0."*; S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling, Global Grid Forum Draft Recommendation, 6/27/2003.

[14] IBM Grid Computing – *"What is grid computing"*, URL: http://www-1.ibm.com/grid/about_grid/what_is.shtml

[15] The Globus Alliance, URL: http://www.globus.org/

[16] Global Grid Forum, URL: http://www.gridforum.org/

[17] *"A visual tour of Open Grid Services Architecture"*, Jay Unger, Matt Haynos, http://www-106.ibm.com/developerworks/grid/library/gr-visual/

[18] *"On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing"*, Ian Foster and Adriana Iamnitchi, 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), February 2003, Berkeley, CA.

[19] *"Grid Computing for the Masses: An Overview",* Kaizar Amin, Gregor von Laszewski, and Armin R. Mikler, Proceedings of the Second International Workshop on Grid and Cooperative Computing (GCC 2003), December 7-10 2003, Shanghai, China

[20] Kazaa Inc., http://www.kazaa.com/

[21] Napster Inc., http://www.napster.com/

[22] OpenNap project, http://opennap.sourceforge.net/

[23] RFC-Gnutella, http://rfc-gnutella.sourceforge.net/

[24] Pastry project, http://www.research.microsoft.com/~antr/Pastry/

[25] Tapastry project, http://www.cs.berkeley.edu/~ravenben/tapestry/

[26] FreeNet project, http://freenetproject.org/

[27] Chord project, http://www.pdos.lcs.mit.edu/chord/

[28] *"Grid computing: What are the key components?"*, Bart Jacob, http://www-106.ibm.com/developerworks/library/gr-overview/

[29] WS GRAM Documentation, http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/index.html

[30] Information Services/MDS, http://www-unix.globus.org/toolkit/docs/3.2/infosvcs/ws/index.html

[31] *"Load Balancing in Structured P2P Systems"*, Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, Ion Stoica, 2003

[32] *"A Framework for Structured Peer-to-Peer Overlay Networks"*, Luc Onana Alima, Ali Ghodsi, Seif Haridi, In LNCS volume 3267 of the post-proceedings of the Global Computing 2004 (pp. 223-250), Springer-Verlag

[33] *"Multicast in DKS(N, k, f) Overlay Networks"*, Luc Onana Alima, Ali Ghodsi, Per Brand, Seif Haridi, In Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS'2003), Springer-Verlag, Berlin, 2004