5-22-2018 10:30 AM

Resource Brokering in Grid Computing

Adrian T. Bienkowski
The University of Western Ontario

Supervisor
PROF. HAMADA H. GHENNIWA
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of Engineering Science
© Adrian T. Bienkowski 2018

Abstract

Grid Computing has emerged in the academia and evolved towards the bases of what is currently known as Cloud Computing and Internet of Things (IoT). The vast collection of resources that provide the nature for Grid Computing environment is very complex; multiple administrative domains control access and set policies to the shared computing resources. It is a decentralized environment with geographically distributed computing and storage resources, where each computing resource can be modeled as an autonomous computing entity, yet collectively can work together. This is a class of Cooperative Distributed Systems (CDS). We extend this by applying characteristic of open environments to create a foundation for the next generation of computing platform where entities are free to join a computing environment to provide capabilities and take part as a collective in solving complex problems beyond the capability of a single entity.

This thesis is focused on modeling "Computing" as a collective performance of individual autonomous fundamental computing elements interconnected in a "Grid" open environment structure. Each computing element is a node in the Grid. All nodes are interconnected through the "Grid" edges. Resource allocation is done at the edges of the "Grid" where the connected nodes are simply used to perform computation.

The analysis put forward in this thesis identifies Grid Computing as a form of computing that occurs at the resource level. The proposed solution, coupled with advancements in technology and evolution of new computing paradigms, sets a new direction for grid computing research. The approach here is a leap forward with the well-defined set of requirements and specifications based on open issues with the focus on autonomy, adaptability and interdependency. The proposed approach examines current model for Grid Protocol Architecture and proposes an extension that addresses the open issues in the diverged set of solutions that have been created.

Keywords

autonomous computing, grid computing, computing platform, resource brokering, agent oriented, grid protocol architecture, cir-agent, JADE

Acknowledgments

Foremost, I would like to thank my supervisor Prof. Hamada H. Ghenniwa for his support at many levels, his belief and perseverance for me to finish the work. It has been a great pleasure working with him and the CDS group on what is to become the next evolution in computing, with new software engineering paradigms and methodologies.

In the CDS group, it has been a pleasure working with Ra'afat, Wafa, Afshan and Ali, where we applied our collaborative research and ideas to practical real-world implementation scenarios.

And, last but not least, to my Wife for her daily support over the many years, and my kids for putting up with my availability.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction

This chapter introduces the field of Grid Computing and presents the fundamental concepts that are applied to the remaining parts of the thesis. The focus of the introduction is to bring an understanding of the field in general and to outline some critical issues in the existing architecture of grid computing environments.

1.1 Grid Computing

The concept of Grid Computing emerged around the mid-70s. The advancement in network technologies by the early 80s, coupled with the idea of implementing remote computing and making that available to a more substantial number of users became a novelty among educational institutions. Grid Computing as a computing paradigm was born. The idea of computing as a grid was borrowed from the electrical power grid model, where the ubiquitous access and use of the Power Grid, was and is part of everyday life. It took the Power Grid model almost one century to become what it is today, an essential and seamless part of our lives. Ubiquitous access is a natural extension and a form of computing, where ubiquitous computing defines the state as expected by the end user and grid provides the architecture as well as the means to deliver that state. There are some fundamental requirements such as usability by the end user, interoperability between devices connecting to the grid, continuous availability and reliability of supply, that have made the power grid something we take for granted in everyday use [58].

The Grid Computing research community has been working on moving the Grid Computing field towards a similar goal, by borrowing and applying fundamentals from the Power Grid model over the past three decades. Although the fundamental concepts seem to be the same, such as a computation as a flow of information over an extensive, Internet size collection of geographically distributed resources, the details of the implementation are much more complex. It is necessary to understand the nature of the problem, its roots and fundamental principles so that the analogy between the electrical grid and computing grid can show how the evolution of the Grid Computing field is still much ahead of its predecessor. Grid computing is also described as an extension of the distributed computing field but at a much large scale [33][54]. Although this seems very natural, the requirements of the distributed computing infrastructure are much different, and such reference hinders its growth. Mark Weiser has coined the term “ubiquitous computing” in the late 1980s, where he defined ubiquitous computing to be the invisible computing that is all around us. A design that is invisible makes the adoption vast. Furthermore, he identified the most significant challenges related to the integration of human factors, computer science, engineering, and social sciences [64].

Some of the reviewed papers, point out the lack of an accepted definition [33][42][54][58]. There is a standard part of the definition of Grid Computing that has repeatedly been quoted. The description related explicitly to VO (Virtual Organizations) is what in my view is pushing Grid Computing to focus on functionality not related to its core. At the core of Grid Computing is simply the availability of computing, where an application can utilize the computing capabilities available to the user. It is critical to stick to the principles of what defines a “grid.” When we borrow the concept of the grid from the Electrical Power Grid, the focus should be on the core fundamentals. However, Grid Computing definition is about connecting heterogeneous, geographically distributed computing resources under the control of different organizations. While a majority of papers extends that definition with “Virtual Organizations,” where VO (Virtual Organization) is defined as a collection of resources and resource very loosely defined to be anything [33]. Grid computing can be focused merely is on large-scale resource sharing, where the resources classification is at the fundamental level, CPU, Hard Disk, and RAM. These three primary concepts make up the essential core of a computing machine. Such fundamental concepts of computing, taken to a large scale of Internet size, create a new set of problems that the field of Grid Computing is attempting to solve. At such a scale the problem becomes around the interoperability, security and access control of geographically distributed computing resources, where the means of communication is unreliable, which is the nature of open networks[33].

In an attempt to define what grid computing is for the purpose of this research, let us outline some key fundamental principles present in the existing grids.

A computing grid

- provides
  - coordinated resource sharing and problem-solving capacity [33][58]
- coordinates
  - resources that are not subject to centralized control [29][58]
- enables
  - service or resource sharing between organizations [52][58].
- constitutes
  - sharing of computer power and data storage capacity over the Internet [58]

The thesis focus is on extracting the fundamental concepts, which build the principle foundation for the definition of grid computing; modeled initially from the Electric Power Grid. Grid computing is a computing environment that is distributed and decentralized. The grid is built from a heterogeneous collection of geographically dispersed computing resources without centralized control. Organizations, which participate as part of the "grid" and offer their resources, are a synonym to the power companies that are all interconnected and supply the power to the grid for end users, being other companies or individual homes. Grid computing in my view is the basis to enable ubiquitous computing which is becoming available all around us [33][56].

Grid computing is also compared with the Internet as related to its size and complexity of the existing infrastructure, the number of connected machines as well as the machines heterogeneous nature. Such fundamentals are essential to note here; this stresses the importance of the role Internet plays in grid computing. Besides the fact that the Internet is the vehicle to enable grid computing across geographically distributed computing resources, it also shares some of the characteristics of the grid". The Internet is an enabler for users to share resources in the form of artifacts such as text and multimedia. The ability to share the artifacts is made possible through the use of standard protocols and interfaces. As well the resources being the shared artifacts on the Internet are dynamic and not subject to a centralized control [33].

One of the essential differences between the Internet and grid computing is the starting point. The Internet has been developed from scratch and overtime during the development, innovation of new technologies and approaches kept replacing existing technologies, which showed to be inadequate for the large scale. This approach, although initially slow, helped to build what we know today as the Internet. Grid computing, on the other hand, had a significant association with distributed computing and hardened network infrastructure. Furthermore, Grid computing had a tremendous vision of what it should be; yet it was built and constrained by the existing mindset of the contributors and applications within academia. The form of implementation of Grid computing in academia has created an environment, where the direction and the developed features have pushed Grid computing away from the original idea, to make computing available to business and people in the same way electricity is today. It seems that critical properties and requirements are still not clear or agreed upon [58].

Scaling and sharing are also standard features of the Internet and the Grid. It is a well-known fact that the Internet has well-established technologies that scale well with the number of users and available bandwidth. However, such claims cannot be made for grid computing. Scalability is a vital issue for widespread use of grid technologies [58]. The key aspects that contributed to the success of the Internet are the reliable infrastructure and relatively low cost for users [30]. The use of standards and the process of creating standards during the Internet evolution profoundly contributed to the reliable and inexpensive equipment that makes up the core foundation and enables the fast and reliable communication between users and resources. It is undeniable to note here that the Internet is an open environment, where numerous diverse systems exist and are connected to serve a specific purpose or functionality.

Grid-computing environments are typically constructed by connecting, either physically or through the Internet, geographically distributed pools of resources for the purpose of sharing. The physical infrastructure that is built out to form a grid-computing environment in itself creates characteristics that can be classified as quality factors, such as, scalability, dynamicity, adaptability and autonomy, as well as the inherited characteristics from the nature of the grid infrastructure and its participants. The inherited characteristics can be broken down into resources and their geographical location as well as policies that come with that ownership. With the ownership of different domains comes administrative aspect, which requires a level of autonomy. The independence of various resource owners needs to be preserved, as well as usage policies. As the grid is built out and expands to millions of resources, the problem of scalability and performance at that scale requires particular attention to performance degradation as the number of resources in the Grid increases. Applications that are written to utilize these resources also need to be designed with latency and bandwidth considerations. Considering that a computing grid is a collection of geographically distributed computing resources, under the control of different organizations; resource availability is not predictable, rather a failure of a resource shall be part of the design of any application, yet the grid is required to be adaptable to its dynamic pull of resources.

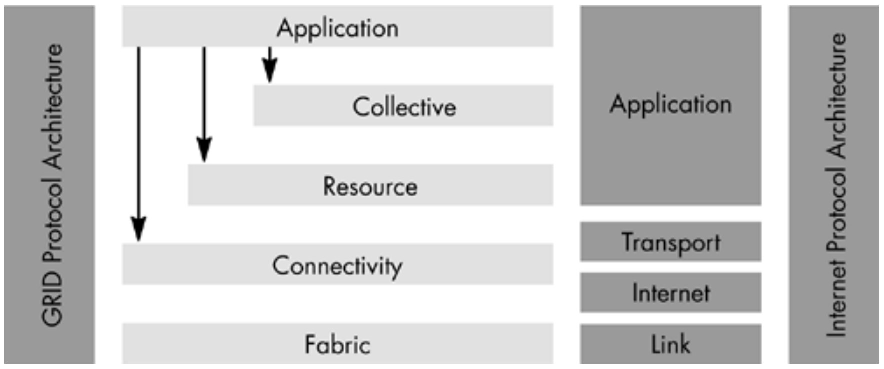

Figure 1-1: Grid Protocol Architecture

Examining the Grid protocol architecture, as shown in Figure 1-1, that has been developed to address key challenges in the existing grid-computing infrastructure will help to understand the current state of the research area.

The Fabric layer of the Grid Protocol Architecture defines the required protocols to support functions related to the resource itself that is being shared [9]. This type of technology supports the capabilities of the resource itself; for example, a distributed file system capability can be supported by protocols such as NFS or HFS. Typically there are no specific requirements for protocol implementation at this level since the implementation is particular to the technology and related capabilities of the resources itself. There is, however, best practices recommended for interfacing with that resource when connected to the Grid. These interfaces are of two parts, one related to the ability for querying information about the state of the resource and two about the ability to manage the resource [9]. The latter part will be expanded later in Chapter 3, where we look more closely at the autonomy of the resource.

The one level above the Fabric is the Connectivity layer. The Connectivity layer is expected to work with any of the transport protocols to provide secure access to the connected resources. This layer is also managing all access control and holds the responsibility for establishing a secure connection for services built and accessed by the upper layers of the architecture stack. The secure connection and access can be facilitated through the use of single-sign-on or delegation type services, where user permissions or a subset of the credentials are relayed to the connected resources. Other security methods, more specific to a group of resources under the control of a particular organization, can include Kerberos, Windows Security, or security of other operating systems. This approach helps to keep the common functionality and the heterogeneous nature of the resource connectivity into a single layer that deals with different implementations at the resource. Security in the grid-computing infrastructure is one of the key foundational constraints. Grid computing infrastructure is composed of computing resources under the management of different organizations. Providing secure access to those resources is a key requirement from both perspectives, ensuring the privacy and security for the user of the resources, and protecting the organizations that make the resource available from unauthorized access. Security service provided by the computing grid also includes data encryption.

Above the Connectivity layer, the Grid protocol architecture defines the Resource Layer. The responsibility of the Resource layer is captured in the classification of two sets of protocols, information protocols and management protocols. The Resource layer uses secure connectivity provided by the Connectivity layer to access and control individual resources that are connected to the Grid. Information protocols are used to query the resource capabilities, configuration and usage policies, as well to get the resource state for monitoring activities. The management protocols allow for more fine-grained control over the resource, yet it is recommended that these protocols should focus on functionality provided by the resource.

The Collective layer gets into the management of resources as a group and takes into account a global state of a pool of resources, unlike the Resource layer where the resources were treated individually and in isolation. The protocols implemented at the layer deal with sharing-type behaviours, such as discovery, scheduling, brokering, monitoring, and diagnostics. The Collective layer allows for applications to utilize a pool of resources, as well as make it possible to build middleware like functionality that exposes all the features mentioned above. Globus ToolkitTM is such a middleware project that will be discussed in Chapter 2. Grid middleware provides essential services for remote process management, access to storage, registration, and user-level security. Grid middleware offers foundation and supporting functionality for application development, programming tools, resource management features and execution of tasks on a global set of available resources. Grid applications and portals are the user-facing part of the grid ecosystem and typically developed using a grid-enabled language. This is a significant contributor to the slow adoption of grid computing as a generic computing platform and is one of the challenges to be discussed in a later part of the thesis. Typically the core grid computing middleware only provides local scheduler functionality and the ability to submit jobs for execution at that level. In such a case, just jobs that have no interdependencies can be sent for execution on the selected set of resources. However, a typical use of grid computing resources requires a much more complex interdependency and conflict resolution. Runtime requirements can change based on the context of the application.

Scheduling is still a very complex problem and as such typically is not part of the core grid middleware. On the other hand data management is part of the core services. A very well-defined set of standards and security protocols are implemented in the Connectivity layer and made available to the services built on top of it. Data management related features provide the ability for the application to have the Grid manage its data and as the application's data is moved through the Grid system to be available to the application during run-time, regardless of the size or geographical location. The grid system has the capability to securely and effectively make the data available to the application, based on the application needs and wants.

Resource management and job management are also fundamental features provided by the core grid middleware to support the local scheduler and execution of the jobs on selected grid resources. One of the very well established grid resource management systems is GRAM(Globus Resource Allocation Manager). It provides the required functionality to execute jobs at the local grid level and support the life cycle of the job execution, starting with the resource allocation, reallocation on failure, check status and deliver results upon completion [42]. Resource management plays a fundamental role in Grid Computing. At the integration level, the ability to discover, allocate, monitor and manage the utilization of grid resources and their capabilities to effectively and efficiently meet global qualities of service. In addition to the integration, resource management allows for a consistent way in which these resources are accessed and managed by services built on top of this layer. The definition of resource management is commonly focused on the process of matching various types of capabilities, arranging for their use and monitoring the state and progress of job execution. The problem of resource management is very well resourced in the context of traditional computing systems. In such an environment the system is designed to operate under the assumption of complete control when applied to grid computing [30].

Grid Computing adds another dimension to the resource management problem, which is well studied in traditional computing systems. Solutions exist for many computing environments. However, they are designed to operate in conventional computing environments with the assumption that there is full control of the available resources. This assumption is not correct when applied to Grid. Resource management at the Grid level requires a different set of specifications that must include, resource heterogeneity, different administrations for policy management and control and loss of that control. In Grid computing environments resources are heterogeneous, and as such the work in the resource management software focused on fundamental issues related to heterogeneity, by defining standard protocols [21][22][30] and mechanisms for expressing resource and task requirements [30][53]. The solution to resource management problems found in Grid

Computing is further complicated by the necessity of having the ability to allocate multiple resources concurrently across administrative domains [20][30][32]. As such current research in this area is focused on understanding different policies from different administrative domains, as well resource provider and resource consumer [6][11] [17][30][43].

The emergence of Grid computing and ability to virtualize all levels of infrastructure and computing resources, virtualized service behaviour is becoming indistinguishable from non-virtualized services and becoming the rule rather than the exception. It should also be noted here that characteristics of grid-computing such as virtual organizations, coordinated resource sharing in our view are the basis for the specification for the application layer built on top of the grid infrastructure, to provide additional features for end users. These would be specific to an industry, and such keeping the focus on the fundamentals allows for a more generic solution.

1.2 Scope of the Thesis

The scope of the thesis is on the architecture of the Grid in the context of open environment. The key contribution is proposing a novel way of addressing the resource sharing by extending the computing to be at the network level. The extension of the architecture allows for resources to connect to the grid, make their capabilities available to other entities in the environment, and participate in the task assignment and execution of requests. The implementation of the proposed solution is also provided with details of the test environment and key scenarios for running the experiment.

1.3 Organization of the Thesis

The thesis is organized as follows. Chapter 1 introduces Grid computing as a computing paradigm and provides a bit of history and analogy to other well-known concepts, such as the power grid. Chapter 2 is focused on the review of existing literature related to the topic of the thesis. Chapter 3 outlines the proposed solution as the main contribution of the thesis. Chapter 4 presents the architecture for the proposed solution. Chapter 5 captures the implementation and results of running the experiments. Chapter 6 is the summary and conclusions of the thesis.

Chapter 2

2 Literature Review

In open distributed computing environments such as Grid Computing, resources may not be available or known when needed and therefore is not feasible to expect to determine or to keep track of the resources and their capabilities. Although these resources are independently created and administered, they are expected to work together to accomplish individual or collective tasks.

Grid Computing traditionally is considered a closed system, in which only authorized and registered resources can be part of the grid and make themselves available as part of the computing pool of resources. Treating grid computing as open environment adds another level of complexity in which, not only the participating resources can become unavailable due to maintenance or failure, but also new resource can join the grid without the need for manual configuration and provisioning.

The open environment approach creates a very hard problem of “managing” computation in particular resource allocation. Such a problem can be solved in a different way. The focus of this chapter is to look at one of the solutions to this problem, which is Brokering, and how others have been trying to address the problem of resource allocation.

2.1 Resource Allocation in Grid Computing

Typical grid-computing installations typically depend on the business needs and application types. The essential computational elements are distributed over a “grid” like structure and interconnected through the edges of the “Grid”. The overall computation “emerges” through the nodes and the edges of the Grid. Unlike traditional computation, where the computing occurs within the nodes, the connectivity is treated as a communication path only. Allocation of computing resources is focused on matching capability, where scheduling takes into account the resource allocation and specific availability of the resource as related to its capabilities.

Tremendous effort is devoted to proposing various algorithms to determine an optimal computation schedule based on the assumption that sufficiently detailed and up to date knowledge of the systems state is available to a single entity (usually called the Metascheduler) [33][42][55][58]. Although such approaches provide efficient means of optimal utilization of the existing resources, they are deemed to be not scalable when dealing with large numbers of machines. Maintaining a global view of the system becomes prohibitively expensive, and additionally, the unreliable networks might even make the scheduling process unattainable.

A different set of solutions to resource allocation and scheduling has also been proposed in the more recent literature. Conceptually, the economic-based approach is autonomous and decentralized to address specifically the needs of a large grid and peer-to-peer platforms [70]. One shortcoming of this approach in the implementation is the fact that it requires a centralized entity to implement and perform economic based functions. Such functions as a marketplace or auctioneer where seal-bid or second price actions can take place.

In open environments, entities need to locate and interact with others who possess the capabilities to achieve a particular goal. The explosion of the grid and cloud computing paradigm, has created, theoretically, an infinite number of available computing resources, for each with different requirements and objectives. However, fulfilling a resource request may go beyond the capability of the individual entities, this is known as the capability-interdependency problem [36]. To overcome this problem in the Grid, different coordination structures and mechanisms that imply various requirements and protocols for interoperability and interaction were proposed. Within this context, the structure refers to the patterns of communication amongst involved participants (for example, brokering, matchmaking or facilitation), whereas the mechanisms define the coordinated control and the interaction protocols.

Traditional resource allocation based approaches in the Grid are to keep the scheduling entities (brokers and/or schedulers) outside the boundaries of the core grid-computing environment. In the following section, a few of the existing solutions are reviewed with the focus on the problems each solution is trying to address.

2.2 Resource Allocation Classification

Resource allocation is a process through which available resources are assigned tasks for execution. The method of resource allocation in grid computing is much more complex, with a large set of data points and multiple levels of requirements this becomes an NP-hard problem, where the exact solution can not be computed in polynomial time. Solutions to this problem typically take a heuristic approach, however, the implementation of the solution can be classified into three categories: centralized, decentralized and distributed.

The existing typical approaches manage the resource allocation from the resource consumer job to the resource providers participating in the grid-computing environment. The traditional solutions focus on handling the fact that most user tasks deal with massive data processing requirements. Applications written to perform a typical job expect that a particular set of input data is available via a standard POSIX (Portable Operating System Interface) call. Besides, these applications produce data as a result of the computation performed and required storage facilities for the output of the execution. The input and output data is expected to be transferred between the computing nodes to make it available to the computing tasks and has a potential to reach several gigabytes in size. Perhaps, the most challenging task of the Grid is thus not just to perform resource matching for job execution, but to ensure that the necessary input is staged-in, and output is appropriately staged-out[26].

A solution required for implementing a grid broker, as being the gatekeeper and providing access to the grid resource, can be divided into three major classes. A centralized approach, where the brokering service is responsible for processing and managing all submitted job requests. In a centralized architecture, the broker can quickly become a bottleneck with performance issues and a single point of failure. Although there are different techniques to address the single point of failure issue, performance issues are however much harder to solve due to a large number of consumers and providers sharing network access to use scheduling functionality. In addition, the centralized approach has the potential to produce optimal schedules, as it has full knowledge of the resources and job requests on the grid.

In the distributed class of broker, the use of a distributed architecture is mainly to address scalability and availability issues, and this makes the grid environment fault-tolerant. However, this type of architecture creates a problem for the scheduling component where only partial information is available to each instance of the distributed broker. A typical approach that has been presented in papers dealing with this class of brokers is a hybrid solution [27]. In the third class of brokers is the decentralized class, where the control of making a scheduling decision is not up to the broker itself. The broker plays a more orchestrating role. A decentralized broker can also be distributed. However, the decentralized notion is referring to the decision-making process. In such a case, the scheduling decision is up to all parties informed and a level of cooperation and coordination is needed to arrive at the scheduling decision collectively.

2.2.1 Centralized Class

The centralized approach to resource brokering does provide the simplest solution to address the distributed nature of grid resources. This type of design allows for a more straightforward approach to solving the scheduling problem, where all information about the system is available to a single entity that manages the resource allocation and assignment. Although the solution becomes "simple" it is not adequate. Even though, the issues that this approach exposes can be addressed through various performance and scaling techniques, the raw size of grid computing environments makes this approach not feasible.

2.2.2 Distributed Class

In the distributed class of solutions, the problem of resource allocation and assignment is solved in two ways. One approach is to move part of the computation into the client side processing, where part of the resource allocation and matching happens. In this approach, the brokering layer of the solution is distributed between the client side tools and the Grid. The second approach in this class is where the resource broker implementation is itself distributed and therefore the distributed computing provides higher throughput for improving performance. The distributed application introduces a new set of problems related to the communication between the physically separated computing resources.

2.2.3 Decentralized Class

The decentralized set of solutions are much more complicated, yet best suited for the Grid infrastructure. In this class of solutions, the typical approach is to apply the economic model and free-market type solutions [28][70], where the participants in a grid-computing environment have full control of the resource assignment and allocation. Decentralization is related to the control, where there is no single entity or system that controls the resource allocation in a dictating or commanding approach. Yet, the control is distributed in a way the participants of the solution have full control over accepting or rejecting the proposed solution through a form of negotiation or collaboration through coordination.

2.3 Current Broker Projects

The flowing section describes existing implementations of grid solutions related to resource matching and allocation. Where resource matching is focused on matching the capabilities with job requirements and allocation is related to the ability to execute the job within the requested time frame.

2.3.1 Condor-G

Condor G[69] implements the resource brokering part of the system and integrates with Globus Toolkit, UNICORE, and NorduGrid. Condor-G does not have the functionality to schedule jobs; it is used as the grid-middleware integration that provides the ability of job execution on grid-computing environments. Another extension is the Condor-G Matchmaker that implements the matchmaking of ClassAds. Multiple Condor-G Matchmaker modules can be used at the same time to improve the scalability of the system. However, the Matchmaker module does not understand parallel jobs and customer scheduling algorithms are currently supported. ClassAds is a language for the description of application requirements and specifications. The language structure allows for a custom definition of attributes, yet it's not clear how these custom attributes are used to match with the published information on the resources to the Matchmaker. Matchmaker uses the posted data by the resource and submitted by the users to schedule job execution on the reduced set of matched resources. However, due to the nature of the environment, typically there is no direct communication between the users and the providers of the computing utility. The ability for the custom attributes to be used in this process would require ontology-driven integration and semantic matching capabilities.

2.3.2 NorduGrid Broker

The NorduGrid middleware is built on standard protocols and interfaces with well-known open source software packages such as OpenLDAP[50], OpenSSL[51] and Globus Toolkit version 2 [31][37]. However, some of the features formerly part of the Globus Toolkit have been replaced with custom components[25]. The custom component approach creates a particular and proprietary solution. The proprietary solution causes a diverging effect on the overall grid-computing space. NorduGrid does make use of the Grid Security Infrastructure (GSI) in Globus Toolkit 2. GSI specifies a public-key infrastructure and SSL (TLS) for authenticated and secure encrypted communication. Also, NorduGrid uses Resource Specification Language (RSL[55], to specify resource requirements and job execution information.

On the other hand, the NorduGrid client-side tools, which mainly consist of command line tools for managing jobs, allowing users to submit, monitor, execution, and cancel their jobs. The resource broker in the NorduGrid implementation is part of the job submission tool, **ngsub**. Other client-side tools allow the user to perform other job-related functions, such as job output and get a peek preview of job output and remove job output files from a remote resource [27]. Putting the resource broker and scheduling related functions into the client-side tools creates potential privacy issues. Also, such issues are not raised in the reviewed papers, mainly, in my view due to the fact that this is a proprietary solution and under the control of trusted organizations. Large amounts of data need to be exposed to perform scheduling, in such an approach. The approach is not adequate for large grid under the control of different organizations and raises privacy issues by potentially presenting sensitive information to a 3rd party.

One of the principal concepts introduced, as part of the work delivered by the NorduGrid team is the concept of Total Time to Delivery (TTD) for the application. This data is used by the resource brokering algorithms in the client toolkit to identify the resources that provide the shortest TTD for the application. TTD is the total time the job will take to execute, starting from the user submitting the job to the time when the output files are delivered to the requested destination. TTD includes the time required for transferring input files and executables to the resource, the waiting time for execution of the job, the actual execution time, and the time to transfer the output files to the requested location(s) [27]. It can be easily noted here that there are a lot of moving parts that make up the TTD value. Almost all of the variables require a level of estimation and it is not clear how this estimation determines the accuracy of the estimates and how or if the estimates are expected to be improved over time.

2.3.3 Nimrod/G

The Nimrod system is a tool that manages the execution of parametric studies [2]. It allows the domain experts to create parametric experiments through the use of a simple declarative parametric modelling language[4]. Nimrod/G is an extension that allows for executing the created jobs from a permutation of the parameters on a global grid [3]. Nimrog/G implements various resource brokering features including resource discovery and management, scheduling algorithms and dispatching of jobs on grid resources. It is one of the first to introduce economic based scheduling algorithm to support user-defined deadline and budget constraints[13]. The proposed economic model allows regulating the supply and demand of grid resources based on Virtual Organizations (VO), also referred to as virtual enterprises[12]. This approach, however, creates a centralized deployment model, to implement economic based aspects of scheduling. The centralized model is further confirmed in [2], where more detailed features are described, such as resource discovery, resource acquisition, and resource monitoring.

Nimrod/G follows what is known as the hourglass design model. Hourglass architecture is used in the design of the IP based protocols. Hourglass design pattern is very much equivalent to a layered design, with the main difference being that the top and bottom layers are extended with many additional interfaces and adapters that are built to interface with the middle layer. Extensions at the top and bottom layer can happen while the core of the solution remains mainly unchanged. As described in [19], the three core components, the task farming engine, scheduler, and dispatcher, are loosely coupled and provide the core functionality of the broker service. The core components are entirely independent of the bottom layer that can be easily extended to support many different low-level middleware features. Also, this type of design pattern allows keeping the core of the solution independent of the external interfaces. Similarly, to how implementation can be extended to support different grid middleware implementations, the top layer can be extended to support different user interfaces with many additional features.

Nimrod/G has been deployed and integrated with the World Wide Grid (WWG) that spans across five continents. However, it still remains a solution for creating parameter-sweeping applications[19]. One could argue that this type of application is well suited to utilize grid resources, due to its repetitive and task-oriented process.

2.3.4 AppLeS

AppLeS (Application Level Scheduling) takes a slightly different approach to the resource brokering in the grid environment. The research put forward by the team which has developed AppLeS, identified adaptability as the fundamental key feature needed for achieving application performance in grid environments[10]. The key difference here is that this project implements application-level scheduling, and therefore the brokering process, which involved resource discovery, resource selection, schedule generation, schedule selection, application execution, and schedule adaptation, is tightly coupled with the application. AppLeS requires the specific application domain to integrate with the provided libraries to perform the scheduling of the application tasks on the grid resource[11]. Naturally, this approach has some limitations and a difficult entry point for an application to adopt this approach. However, the scheduling algorithm can be very well adapted for the specific needs of the application and its domain, as well the scheduling decisions that take an iterative approach where each iteration through the process uses previously scheduled data as input for following iteration[10]. To improve the adoption of AppLeS methodology with application-level scheduling and the integration of the AppLeS scheduling agent into the AppLeS-enabled application, the team has made available AppLeS Templets available for different application types[10]. The AppLeS Template is a software framework that allows for the application to insert its application-specific logic into the grid scheduler, and create a self-scheduling application for grid-computing environments. The templates have been designed for different application categories such as parameter sweep and master-worker type[10].

2.3.5 GridWay

GridWay is a meta-scheduler, with support for grid middlewares such as Globus Toolkit and gLite. The support for different grid middlewares is done through adapters, with the literature on GridWay broker referring to these adapters as MADs (Middleware Access Drivers). Two additional parts are attached to the GridWay core; the pluggable UI and the schedulers. The logical view of this architecture is similar to that of Nimrod/G, a typical way to abstract the external layers that require change and adaptation, away from the middle core layer, which contains the business specific functionality. Yet, the internals of the core middle layer are much different and focus on the flow of the job request. Different, manager like components provide the integration with external middlewares for things like grid information, job execution, file transfers and job migration [63].

GridWay puts the focus on application performance; this is the primary driver behind their architecture and features related to the core functionality. It is also the main contribution put forward by the team behind the GridWay project. The very valid consideration that network performance has a significant impact application execution. GridWay broker adds essential network characteristics to all states of the job request, including scheduling, migration, and monitoring [63]. The process that includes the network related information is called automatic network-aware meta-scheduling architecture and is capable of adapting to the current status of the environment. One item that is not clear in the current literature is the consideration of the job request itself and its impact on the environment during the future execution of the submitted job. The implemented algorithm uses exponential smoothing to predict the job execution. Exponential smoothing, also called simple or single exponential smoothing (SES) is a method for forecasting data with no trend or seasonal pattern [63].

2.3.6 GRUBER/DI-GRUBER

GRUBER (**G***rid* **R***esource* **U***sage* **B***rok***ER**) main contribution is managing USLA (**U***sage* **S***ervice* **L***evel* **A***greement*) in the grid and grid-like environments, where consumers and resource providers span multiple organizations. GRUBER understands the different aspects of an SLA related to its representation, enforcement, and management of the service agreements [24]. Service level agreement is a contract between the service or resource provider and the consumer of that service or resource. The agreement defines the expected level of service the provider is expected, or in most cases guaranteed, to offer. Based on the USLA the customer of that service is able to assume the availability and performance, and plan its operations accordingly. In addition, the consumer also shall abide by the agreement and use the service or resource in an expected way, and within the boundaries of the agreement. This concept in the context of distributed systems, and more specifically grid-computing, creates a very hard problem to solve for the broker where the resource providers to the Grid are subject to different policies under the control of different virtual organizations [33].

DI-GRUBBER, which is a distributed version and an extension of the GRUBER, to address scalability and performance issues in a large distributed systems, and more specifically, grid environment. DI-GRUBBER adds multiple decision points to the scheduling algorithm and provides the ability to efficiently store, retrieve, and publish USLAs in a grid environment [23].

2.3.7 SPHINX

In the context of grid computing, it is the guardian that protects computing resources and schedules jobs for execution. The project has been used in scheduling complex and data-intensive applications in high energy physics and data mining projects, which required grid computing size environments to execute.

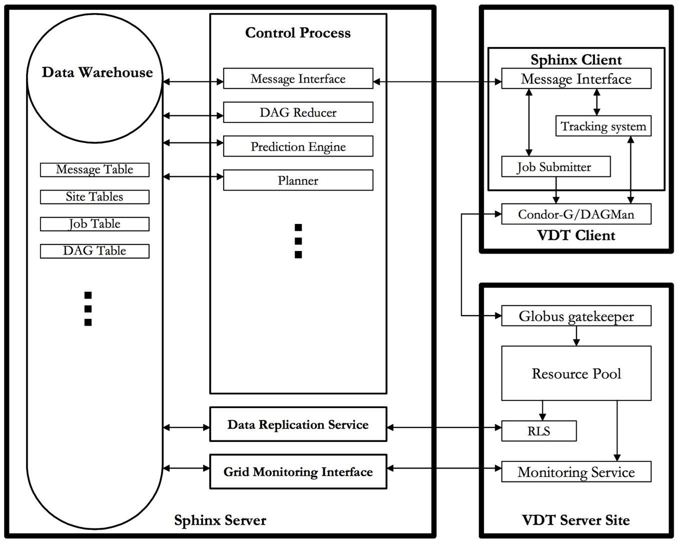

Figure 2-1: The Sphinx Scheduling System [44]

The SPHINX project implements a much different architecture, as shown in Figure 2-1 above, then presented with other brokers in this chapter. For state persistence, it uses a Data Warehouse where information gathered from Data Replication Service and Grid Monitoring Interface is transformed into different informational tables for the control process. The tables in the data warehouse are used for caching information related to cataloguing, job tracking, and the list of resources. The control process, in turn, is used as the central brain and orchestrates the changes in the state of the submitted job. Although the related literature talks about the architecture as client-server, the diagram as shown in Figure 1-1, reflects a server-agent architecture, where the agent is doing the work assigned by the server, and various implementations of the agent take the form of an adaptor to support different local resource management systems. The light-way implementation of this part of the system validates this statement as it is the connection to the external components.

The implementation of the SPHINX broker is based on two kinds of requirements, informational requirements and system requirements. Informational requirements have been captured as the core requirement for scheduling component. It identifies the different sources of information that are needed to perform complex scheduling operations on the grid resource. The system requirements are focused on the efficiency of the scheduling that includes QoS (**Q***uality* **o***f* **S***ervice*), extensibility, customizability, and interoperability [44]. The type of requirements identified here is mainly quality factors and not directly related to the required functionality of the system. These quality factors are also very common to most non-trivial systems and not just limited to large and complex systems such as distributed systems and grid computing environments.

Chapter 3

3 Problem Analysis

Grid computing has emerged in the academia and evolved towards the bases of what is currently known as Cloud Computing and Internet of Things (IoT). The vast collection of resources that provide the environment for Grid Computing is very complex; multiple administrative domains, controlled access and set policies to the shared computing resources. It is a decentralized environment with geographically distributed computing and storage resources. In a grid-computing environment, the computing resources might be heterogeneous [33] where no assumption can be made about the operating system, CPU capabilities, and memory or storage capacity. The assumption is that mapping computing resources, including resource allocation problem, is a collective responsibility given that the computing entities participating in the environment can have full control over accepting or rejecting the allocation. In the traditional approach, the resources are controlled from the upper layers of the Grid Protocol implementation as described in the next section of this chapter.

The research described in the thesis presents Grid Computing as an open environment. Open environment approach introduces a new level of complexity as related to interoperability, adaptability, autonomy, and interdependency. In addition, Grid Computing is viewed in the context of CDS (Cooperative Distributed Systems). CDS is a class of distributed systems where entities have full control over their actions, which includes sharing and control over their capabilities. In such a classification no single entity is capable of solving problems in isolation, and a level of cooperation is required between the entities to accomplish their goals. CDS is a computing paradigm where entities in that environment are expected to collaborate and work together. The size of the environment, by nature being large, provides that entities with overlapping capabilities do exist and are part of that environment. In such a case interdependency is a known problem; more specifically capability-based interdependency creates the problem where a solution to this problem is coordination [36].

3.1 Grid Protocol Architecture Analysis

The Grid Protocol Architecture as shown in Figure 3-1 illustrates a composition of functional layers for implementing grid computing protocols and services. The application layer of the stack has a direct interface to all lower layers. The flexibility of the application layer of the protocol stack, having direct access to multiple lower layers might lead to a complex architecture at the Application, through which it doesn't provide the separation between the application and the supplied capabilities of the environment. This signifies that any service implemented at this layer has the freedom to use and control protocols from the layers below. Therefore, there is no clear separation of concerns when it comes to the use of the protocols and how they are utilized at the application or service built on top of the Grid Protocol stack. Based on that it is open to interpretation and up to the implementation of the Application layer and how these protocols are consumed. In other words, in the Grid Protocol Architecture, both Collective and Resource can be directly modelled as part of the application; shown in Figure 3-1(b).

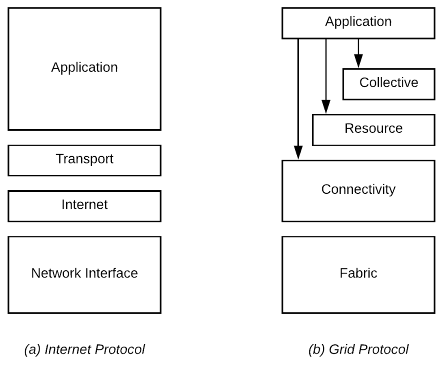

Figure 3-1: Simplified Grid Protocol Architecture

The Fabric layer provides access to the local control has a clear separation from the Connectivity layer and can only be accessed through the supplied secure protocols provided by the Connectivity[33]. Although the protocols are specific to a closed system and can present difficulty in open environments, it is still a good foundation where communication protocols can be extended to support collaboration between entities in the environment. Once we go above the Connectivity layer, things start to break down. From here, the Application has direct access and can become the facilitator that manages the Collective and Resource through Connectivity and therefore provides an open landscape for creating problem-specific solutions that cross the top three layers of the protocol stack. Furthermore, the definition of each Connectivity and Resource layer emphasizes the access control and management of the lower level resources [33]. In an open environment, this becomes problematic since the resources are required to be autonomous, with full control over their capabilities and adaptable to the changes in the environment.

The other aspect of a grid-computing environment, which is only partially addressed in the current Grid Protocol Architecture, is the fact that the environment is composed of a large number of resource providers and resource consumers. In such an environment the level of interaction required to collectively work together to solve problems, requires coordination and cooperation of the participating entities. Yet, the current Grid Protocol Architecture provides, among other things, Information Protocols to query the state, Management Protocols for access and resource control and Directory Services for querying resource name and attributes[33]. Such architecture promotes a top-down management and control approach instead of supporting autonomy and collaboration. The size of the computing grids is considered "large"; Internet-size large. Therefore, the interaction between the consumer and the providers of the computing resources creates a high level of complexity. Furthermore, due to a large number of resource providers, there will be overlapping capabilities between the number of providers and that creates a capability interdependency problem. The nature of such an environment and the fact that the structure of the system is not known at design time, besides the roles of the entities making up the environment, creates an architecture where communication between the entities is a problem. A large number of entities requesting assistance to solve a problem from other entities in the environment do not adhere to a specific architecture, and it creates a many to many relationships, which only materializes at run-time. A solution to this problem is called brokering architecture pattern[46].

Brokering is an architectural pattern based on consumer-provider model for the interaction in a consumer-provider type environment. Consumer-provider is a general problem where large numbers of consumers interact with a large number of providers in order to achieve their goals. Goals can be further divided into objectives based on environment and entity-specific parameters. The consumer-provider solution is an NP-hard problem, and therefore to reduce the complexity, a resource brokering approach is used. The difficulty remains in that we are still dealing with existing systems and an architecture that is guided by the properties of the existing grid infrastructure.

3.2 Open Issues

This section of the thesis will examine and define each of the open issues that will be addressed in the next chapter.

3.2.1 Autonomy

Autonomy is the ability to self-govern, from the Greek meaning independent. Autonomy is a fundamental aspect of agent-oriented architecture, where the design of the system accounts for individual entities to interact and coordinate to solve a common problem. Autonomous entities exist in an environment and perform operations on the environment based on their internal state, their knowledge, beliefs, and intentions. Autonomous entities are designed to deal with the unexpected, where autonomous agents and multi-agent systems can leverage the capabilities of other agents to accomplish a goal; an unexpected state of the world causes a condition where a single agent cannot perform such operations. A software agent is autonomous; it can use its knowledge and the current state of the world to modify the environment and accomplish its goals.

In an open environment, where a large number of consumers and a large number of providers exist, it creates a complex system where interaction between consumers and providers becomes very complex. Entities that exist in such an environment can come and go at will, and are not part of the original architecture. The architecture of such a system changes at run-time. Designing a system using agent-oriented architecture creates a system that is more adaptable to change and inherits the ability to adapt at run-time to unexpected events; such properties make the entities autonomous.

At the time of writing this chapter, there is no literature related to resource brokering in open environments. In order to design a solution for such an environment, one needs to study the characteristics that define an open environment. By definition, open environments are dynamic, where software systems are arranged at run-time to create a solution[72]. This type of system requires interaction and autonomy at the individual component or service level. Autonomy is a property of a software system that allows it to function under dynamic and unknown conditions and therefore a key characteristic of software entities designed for the open environment. The proposed solution will shown the ability to dynamically select a scheduling algorithm, different scheduling algorithms may not be available in the initial design and deployment of the system. However, additional scheduling algorithms can be deployed into the environment as separate and new entities with specific scheduling capabilities. Autonomy yields the required capability for entities to cooperate in an open environment. A particular type of interaction called coordination can provide solutions to problems that previously did not exist. The solution is composed of a newly formed system that can solve the problem, all done at run-time. For example, two autonomous robots exist in a building with different rooms and doors between each room. Each of the robots has different capabilities and the ability to communicate and cooperate, however, they are not designed as part of one system. Now, if a request is made for one robot to be present in a different room and that robot does not have the capability to open the door, in order to move between rooms, it would fail to fulfill the request. However, since another robot in that space exists with the capability to open the door, the goal can be attained by the first robot to have the door opened and moved to the desired location.

The proposed solution applies the agent-oriented architecture approach to create an adaptable architecture where entities can join and leave at will. New entities can join the system with new capabilities to support new ways of assignment of computing resources and schedule those resources to perform tasks. The communication protocol that enables the coordination between the entities allows for entities to cooperate and coordinate their actions. If one entity is not able to perform the required actions to complete its goal, it has the ability to communicate other entities and request assistance.

3.2.2 Adaptability

In the context of a computer system, the ability for computer systems to adapt to a changing environment is typically not a requirement during the design stage of the system. As the computing paradigms have evolved over the past two decades, both system architects and system developers see this as the next evolutionary and essential step in the next phase of evolution [7]. In grid-computing adaptability shall be considered as one of the key, non-functional requirements.

Adaptability is also a fundamental characteristic of an open environment, in which a computer system has the capability to change and adjust its behaviour based on the changes in the environment. Although, adaptability is not an obvious issue in the existing Grid systems, mainly due to the fact that current view of the Grid is a closed system. Yet, the issue has been explored in some recent papers [7][15][48][59][73]. The evolution of the Grid to support an open environment makes adaptability, or the lack thereof a key issue, and is proposed here as part of the solution in this thesis. In such context, adaptability of the participants in the computing environment is a requirement, yet the existing solutions do not take this into consideration.

Most all of the reviewed literature explores Grid Computing as a closed system, where a specific process is required in order to add additional computing or data resources to the existing grid infrastructures. The solution that is proposed to solve this problem is to look at adaptability at two levels. First, it looks at the solution from an open system perspective and requires the use of standard protocols to facilitate the communication between entities participating in the system. Secondly, it defines the protocol at the interaction level of the entities in order to enable efficient coordination. The efficiency of the cooperation of the entities participating in the computing environment is accomplished by introduction of the resource-broker service into the environment.

3.2.3 Interdependency

The word “interdependency” can be analyzed by breaking it down into two parts; "inter" and "dependency". "Inter" meaning between or among, and “dependency” meaning that something is dependent on something else. Therefore, “interdependence” is a situation in which two or more entities depend upon each other. Such a situation can only happen if more than one entity exists in an environment, and the entities have overlapping or dependent capabilities. This is also known as capability-based interdependency problem [36] and it is found in settings where multiple entities of similar and overlapping capabilities exist; such is the grip-computing environment[36]. In such an environment entities need to interact in order to solve their interdependencies and this requires a level of cooperation as shown in the next section 3.6.

Interdependency is a problem due to the overlapping dependence between entities in an environment. There are different interdependency types, and they are classified based on the domain of interaction. In such classification, we find the following types [36].

- *physical* - physical aspects of the environment require the sharing of resources between the entities [36].
- *mental* - can be classified as capabilities, knowledge and interest [36]; this is where entities are dependent on other entities to accomplish their goals.

Further, there is a topology to the interdependency types, which deals with the type of interactions of the entities [36]. For example, in a physical conflict where two entities require the use of a single resource, they will need to interact and share the resource based on the topology of their interdependency.

- *pooled* - which results in the least amount of conflict [67]; entities do not directly depend on each other and instead use a common pool of physical resources.
- *sequential* - when the dependency of one entity is on the output of work done by another entity.
- *reciprocal* - results in the highest potential for conflict [67]; the information flow is required to go both ways.
- *comprehensive* - similar to the above with a greater complexity of interactions [67], resulting in the potential for very high conflict.

The problem becomes more apparent in the context of an open environment, where entities are capable of joining and leaving an environment at will. This creates a dynamic architecture that changes at runtime. The nature of the system creates a problem where entities with overlapping capabilities do exist in the same environment. This creates an interdependency problem with one or more of the topologies stated above. A typical approaches to solving the interdependency problem is through predefined architecture, where specific requirements and specification for system design dictate the relationship between components of the system architecture. In the case of open environments, the system is dynamic, and the interdependency problem exists at runtime. Others try to solve this through a centralized or federated approach and manage the resource capability overlap through a database information system, where the information needs to be updated and kept up to date very frequently, in order to be current and usable.

Interoperability is a problem identified in [33], the solution to interoperability is a standard protocol implementation that allows for interaction between components. Interoperability is a core requirement in open environments, as the protocols need to be known to entities that are not part of the environment and would like to join. Our focus is on open environment where we extend the Grid concept to Internet-size open environment where entities can come and leave at will and system has the ability to adapt. A standard communication protocol allows for interoperability between entities to allow them to interact and coordinate. In the literature review, some of the requirements for resource broker have been collected from different sources. The discussion below will show how all of these requirements have been considered during problem analysis and have been shown to address the issues outlined in the proposed solution.

The proposed solution looks at the interdependency problem from the perspective of interaction, where coordination is the solution to this problem and will be discussed in the next chapter.

3.3 Summary

The analysis in this chapter has shown how the current approach and interpretation of the Grid Protocol Architecture has allowed for a dispersed set of solutions. There are almost as many implementations that provide resource brokering as there are Grid deployments. The complexity of the problem has been identified as an NP-hard type problem for which there are various possible heuristic approaches to provide an optimal scheduling solution. However, this is not the focus of the thesis. The thesis is focused on delivering an architecture solution to the resource-brokering problem by extending the current Grid Protocol Architecture and applying the concept of open environments to the problem. Also in this chapter, we have identified some key open issues that will be addressed in the chapter to follow.

Chapter 4

4 Proposed Solution

The proposed solution is addressing the currently known open issues as identified in Chapter 3.2. The open issues have resulted from simplifying the model of grid-computing and extending the model to include concepts of an open environment. In the mainstream literature on Grid Computing, the traditional approach is a closed environment. However, with the explosion of web-based services, the Internet has become a collection of heterogeneous computing resources, each providing a different function, with partially overlapping capabilities. Although this environment somewhat resembles grid-computing, the fundamental difference is that the Internet is an open environment where resources connect and disconnect, without centralized control. The analysis in the previous chapter outlines the current state and the ability of grid-computing to evolve beyond the traditional view. In today's field of computing, the advancements have put forward a new computing paradigm, and such is cloud computing. Cloud-computing has been built on the longtime promise of "utility computing"[64]. "Utility computing" was coined by Mark Weiser back in 1980, where Grid-computing has fallen short to deliver on such vision; Cloud-computing has provided a computing paradigm as a utility, at various levels, IaaS, PaaS, SaaS, and various other XaaS (anything as a service) variances. Although such implementation does not realize the vision as presented by Mark W., it is a step closer in that direction.

4.1 Model

The Current model of Grid Protocol Architecture as depicted in Figure 1-1(b), is associated with strong assumptions that are not adequate for open environments.

The Grid architecture assumes the application has full exposure and control over the *Collective* and the *Resource* layers. In addition, the application also has full access to the *Connectivity* protocols to manage and control the *Resource* layer, directly or through the *Collective* layer. Although it aims to support flexibility, it led to complexity and different implementations of brokering functionalities. As such, the integration of the application with a resource broker requires custom logic and communication protocol specific to that implementation. This restricts the implementation to support open environment applications. In the proposed solution the aim is to re-define the Grid computing architecture supporting open environment principles, in which the entities making up the environment are not known at design time. This requires entities that exist in such an environment to be supported with the ability to adapt to the change of the environment and interact accordingly. This requires a level of autonomy at the entity level. Each entity needs to have the ability to perform an action on the environment based on its capabilities and the ability to interact cooperatively with other entities in order to achieve the individual or the collective goals.

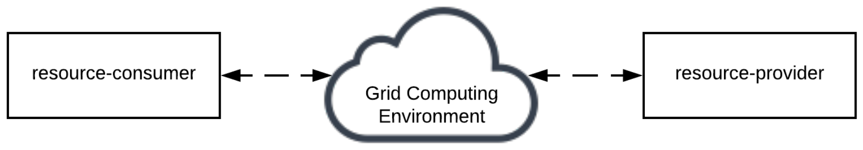

Figure 4-1: Grid meta-model

Applying these concepts to the structure of grid-computing environment creates a meta-model for the system, depicted in Figure 4-1. It classifies the application into resource-consumer and resource-provider. The dashed line in the diagram represents the interaction between the two classes of the entities roles. In the context of the open environment, both resource-consumers and resource-providers are not known in advance. To address this assumption a brokering-based model is proposed. Brokering provides a dynamic interaction among resource consumers and providers that are not necessarily defined in advance. It enables a resource consumer submitting a job request without the need to know about all available resource-providers. Similarly, for the resource-providers, which can participate indirectly for interaction with resource-consumers only known at run-time. A resource-provider is modeled in terms of capabilities within the grid computing. Then through the brokering resource allocation and assignment based on the job requests from consumers and managed the interaction is carried out with the resource providers for their commitment to the job execution.

The resource allocation is based on identifying the adequate providers for the submitted jobs based on predefined criteria.

4.2 Architecture

The proposed brokering-based model captures *Computing* in the form of job's *Capabilities* and *Resource Allocation*, where the application is unaware of the individual *Resources* that execute it's jobs. The application has a view of the Grid as a virtual single computing platform with the required capabilities to perform the requested job.

In a typical grid-computing environment the resources that provide computing capabilities to the grid, connect through the network to a centralized service, which is responsible for scheduling the job of a particular type to execute. Different resource brokers capable of executing jobs of different types have been outlined in Chapter 2. Such architecture uses the network to facilitate the communication between resource providers and other systems making up the grid-computing environment.

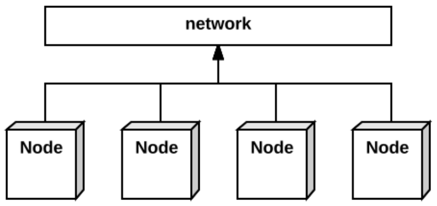

Figure 4-2: Traditional distributed system

The proposed solution is extending the network to include the job-resource allocation and scheduling. This model transforms the network from merely a communication-based layer to a foundation of Grid-based Computing platform. In addition, it is naturally adequate for open environment applications, where there is no centralized control over resource-providers and resource-consumers with the ability to join and leave the computing environment dynamically.

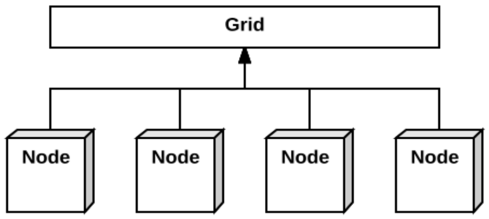

Figure 4-3: Proposed Architecture

The proposed architecture is based on the brokering model. We identified four distinct functional areas of the resource broker, job handling, resource matching, scheduling and job execution, as shown in Figure 4-4.

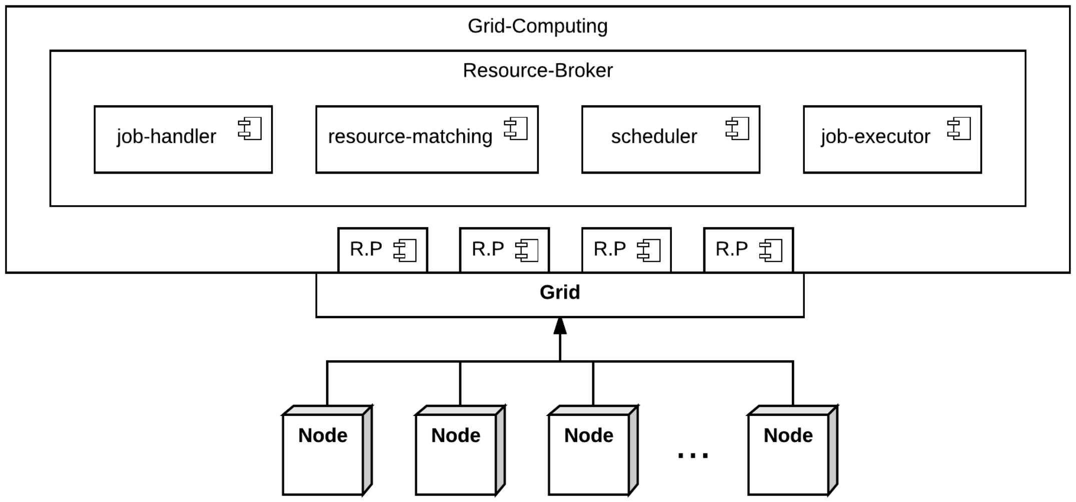

Figure 4-4: Proposed Architecture with Resource-Brokering

A resource-provider (R.P.) modeled as a node is an autonomous entity represented as a computing node, possessing computing resources is modeled as autonomous entities, which have control over their computing resources, which have control over their computing resources and make their computing capabilities available to the grid computing environment.

4.2.1 Proposed Grid Protocol Architecture

Existing Grid Protocol architecture as shown in Figure 4-5 Grid Protocol, allows Application layer implementations to control all aspects of the Grid. This approach allocates all the control at the top layer of the protocol architecture and therefore creating a centralized control mechanism and structure in the Grid environment. Further, this type of architecture reduces the ability for resources to be autonomous, where autonomy is one of the key requirements for entities in an open environment.

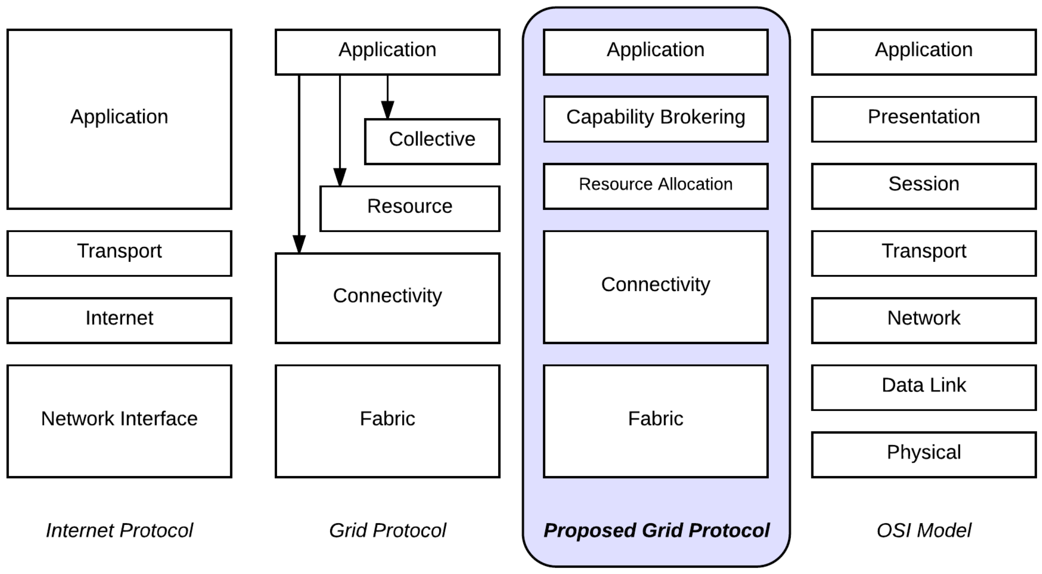

Figure 4-5: Grid Protocol Architecture (*current and proposed)*

The proposed Grid Protocol Architecture, Figure 4-5, provides the Grid with a better separation and controlled access to the Fabric of the Grid. The Application layer has direct access the supplied capabilities of individual or group of resource that make up the Fabric. Using a secure communication over the Connectivity layer, resource or groups of resources have the ability to register their capabilities. The Resource Registration layer provides the required protocols to carry on the brokering facilitation as an integral part of the Grid. Unlike the traditional approaches where brokering can be implemented as a separate application at the high level of the stack.

4.3 Grid Computing Entities

The autonomous entities in the Grid are modeled as software agents. Agents architecture is a composition of knowledge and capability. Each component in the resource-broker service, normally job submission, resource matching, scheduling and execution, can be modeled as a software entity. The capabilities of the entity depend on its designated function. The system in this context is known based on the roles and responsibilities required to perform the brokering service. This approach allows for separation of concerns and well defined coordination pattern among the participating entities. For example, job-handling agent has responsibilities of accepting a job request, validating all required parameters and passing the job to the resource-matching. The resource-matching agent can be performed by multiple and different agents based on the parameters supplied in the job definition. New agents can be added into the system without changing the architecture, yet providing new capabilities that are not known at the initial design of the system.

Additionally, a resource-matching agent interacts with scheduling agent to assign the execution of a specific job to a specific resource at a specific time. Here the type of scheduling agent is selected at run-time based on the scheduling capabilities required by the submitted job. Furthermore, multiple agents representing the resource-providers can manage the job execution with the selected resource-provider as computing entity. These capabilities can interface with different grid middleware.

4.4 Computing Platform

The computing environment that defines the platform naturally comes with a level of abstraction and constraints related to supporting a specific type of applications. In the context of a “Grid”, the computing platform has to provide not only accessibility to the computing infrastructure, but the computing infrastructure needs to expose capabilities with four key requirements. The application should not directly manage computing resources nor the assignment. The entity’s autonomy and self-management of the exposed capabilities, coordination between the entities to perform the required task, and adaptability of the platform to self-adjust when resources become unavailable to perform the required computation are the fundamental guidelines that drive the platform's requirements.

The proposed solution creates an architecture where multiple agents exist and coordinate their actions to accomplish a common goal. It views coordination as the solution to the interdependency problem with the specific focus on capability-based interdependency. The platform supports the following primary aspects of open environment applications.

4.4.1 Adaptability and Autonomy

The agent-based model provides the platform with a well-defined structure, as related to software entities that make up the computing environment. The entity structure enforces the required guidelines to support open environment applications. The open environment requires that entities are autonomous and can interact with other entities in that environment. Similarly, adaptability is required for the entities to interact and to solve new problems, which did not exist at design time.

The proposed brokering-based Grid protocol architecture enables the application of a resource-consumer to adapt to the changes of the environment in terms of resource-providers.

4.4.2 Coordination and Cooperation

Coordination provides a structure and mechanism for agents to interact cooperatively. Cooperation can take a number of different forms; on one end being totally of self-interest and the other end being a total sacrifice. In this work, our focus is on cooperation for a mutual benefit to accomplish a common goal, where the coordination is the solution to the interdependency problem in a cooperative distributed system where agents exist in an open environment. The coordination of different resource-providers in an open environment is addressed at the application layer. Coordination is viewed here as the solution to the interdependency problem [36].

The concept of coordination is not unique to computer systems. A significant amount of research is done on the topic in biology to investigate how different living entities coordinate. An interdisciplinary theory of coordination was developed by Malone[45].

The study was done across multiple disciplines, including among others, computer science, organization theory, operations research, economics, linguistics, and psychology. Malone defines coordination as managing dependencies between activities. The dependencies in the context of the thesis are between overlapping capabilities. Similarly, [33] also outlines the interdependence between resources at the lower level of the grid infrastructure and the support for advanced reservations or advanced scheduling.

Coordination has been a well-researched topic in a number of different disciplines; however, researchers often presented coordination as a problem. In the context of distributed systems specifically, the type of distributed systems that are part of this thesis cooperative distributed systems, coordination is a key component. In our research group, the focus is on the special class of distributed systems, called CDS (Cooperative Distributed Systems) where entities exist and cooperate. In such system, we view coordination, as a key solution to enable the cooperation between entities to resolve problems, where capability based interdependency is the problem.

Chapter 5

5 Grid Computing: Framework & Implementation

In the previous chapter, the architecture for the proposed solution shows how extending the network with connected resources can bring computing to the network level and create a grid-computing environment where the computing happens on the network. This is an extension of the phrase, "*Network is the computer*" coined by John Gage from Sun Microsystems back in the mid '90's. This vision statement has become more accurate over time. At the time of the vision, single computing nodes or clusters of computers which make up distributed systems were not enough to solve the complex problems. The statement recognized that computing needs to happen beyond the physical limitations of the computing node or a centralized management layer of a distributed system. In a similar way, grid computing needs to go beyond the boundaries of the computing nodes, and bring computing to the network layer, where the computing nodes are a simple resource with capabilities to participate in the execution.

The design and implementation of the proposed solution brings about a number of challenges. Each of the challenges is outlined in the following section of this chapter. Section 5.1 discusses the design requirements and choice of technologies used in the implementation. Section 5.2 outlines the design and implementation of the resource-broker. Section 5.3 and 5.4 contain design and implementation of the resource-provider and resource-consumer respectively.

5.1 Computational Model

To model the computation as described in Chapter 4, I break it down into two main parts, Capability Brokering and Resource Allocation. The system represents a grid-computing environment that is an open environment, where entities have the ability to join and leave the environment at will. Open environment characteristics require participating entities to adapt to the changing environment, inhibit a level of autonomy and have the ability to coordinate. The characteristics of an open environment require a different approach to designing a system for the proposed solution. In the traditional OO paradigm, the system functionality is described in terms of interoperating objects. The principles of this design approach focus on abstraction, encapsulation and modularity. In such a case it does not provide a way to directly address the characteristics of the open environment, where adaptability, autonomy and interdependency are the key requirements that need to be addressed.

- *Adaptability* – not all the participants of the system are known at design time; this requires the known participants to be able to adapt to the environment.
- *Autonomy* – there is no centralized control; in an open environment each entity has a self-interest and control over its resources.
- *Interdependency* – the fact that a large number of entities do exist in the environment and entities are able to join and leave the environment, and create a capability interdependency problem; the solution to such a problem is coordination between the entities.

Considering that we are dealing with self-interested entities in an open environment; an agent-based model is a better choice for the design approach. Agent-oriented is the next generation for software engineering paradigms and computer modeling of open environments and autonomous entities. Agent-oriented design provides the following principles as part of the design principles: autonomy, adaptability and coordination.

Based on the requirements the choice for the implementation of the proposed solution is an agent-oriented approach.

5.2 Brokering Architecture

The use of agent technology here is essential, as it not only maps directly to the fundamental aspects of the existing Grid computing environment, it also addresses key aspects of the requirements. In the agent-oriented design process, the focus is on the way the agents in the system cooperate to accomplish system-level goals. The goal of the process is to transform the high-level concepts from the analysis into sufficiently low-level abstractions [71]. At which point traditional design techniques such as OO can be applied and used to implement the software system. In the analysis stage we develop the understanding of the system in the context of its organization. This process will focus on identifying the roles of the entities in the system and the required interaction between them. The outcome is expected to deliver autonomous self-interested aspects that are a natural fit for the Grid environment. In the Grid environment, there is no centralized entity to manage the overall dynamics and interaction. Further, we introduce another layer of abstraction at the participant level. The Grid environment consists of resource consumers and resource providers that utilize the Grid to share their capabilities. We abstract both the resource consumer and providers with our agent technology in order to bring computing to the network level and decouple the computing infrastructure from the interaction of different components, see Figure 5-1.

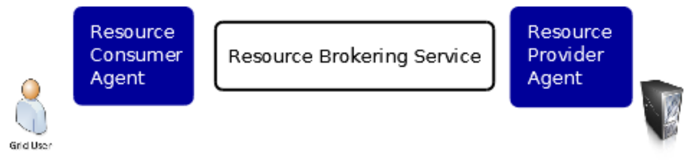

Figure 5-1: Agent-oriented design

The approach to design of the system is focus on defining key interfaces between each component. The first part looks at interfaces required in order to abstract the lower parts from concrete implementation. Following that is the look at each known entity of the environment and the role they play. This looks at the high-level view of the resource-broker itself and provides a detailed look at the components making up the resource-broker. Further, we examine the overall system and the complexity of the interaction between the different entities, as well the similarities in the design of each entity as an agent.

The design for the system takes into consideration the fact that different technologies and some off-the-shelf solutions will be used in the implementation. This requires the use of the following principles in the design of the system, where a layered design of the solution will provide abstraction from the different technologies and we use separation of concern design pattern to provide a clear interface to each of the concrete system functions. Creating an interface for each layer and programming to that interface will provide the ability to switch to a different implementation of a specific component at run-time.

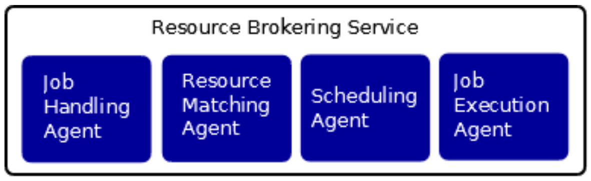

Figure 5-2: Resource-Broker Architecture

The known entities making up the system, as shown in Figure 5-1, are grouped into three categories, resource-consumer, resource-provider, and resource-broker. Furthermore, resource-broker is subdivided into four components, as shown in Figure 5-2, specific to the required functionality provided by the brokering service.

5.2.1 Agent Model

CIR-Agent is the agent model used in the CDS group, and it has been chosen for the implementation of the proposed solution as part of the work in this thesis.

5.2.2 CIR-Agent Interface

CIR-Agent interface has been created in order to decouple the application from the technology used in the implementation. A CIRAgentJADE class has been created as a separate project, which implements the CIRAgent interface and hides any platform dependent code. The application layer then uses this interface and provides a well decoupled design, where the underlying platform, JADE, can be removed and a new class for the new platform which implements the CIRAgent interface is put into the classpath and the software component will keep working without recompiling.

5.2.3 Action Interface

Action interface was created in order to have a functional implementation with the highest level of autonomy. An Action interface provides a single action method for execution of the given functionality. The Action interface is utilized in the CIRAgent implementation to perform work on the environment with the highest level of autonomy.

The Action interface also provides a way to set pre-conditions and post-conditions which are utilized by the problem solver within CIRAgent implementation. The problem-solver is able to compute a solution for a particular agent's goal, based on this relationship.

5.2.4 Agent Interaction

The third part of the implementation is the interaction between the different components. The components which make up the system are the resource consumers and resource providers, as well the components that make up the broker service itself. As mentioned in the previous section of this chapter, the assumption that the broker service is a trusted entity and provides the highest available PPL value for the consumer. The interaction between broker service and external entities can be as follows. Resource consumer uses the standard ControlNET protocol to submit a job request to the broker service using the CFP message type. Within the message the job description is attached. The currently supported job description language is JSDL (Job Specification and Description Language), a standard proposed by the Open Grid Forum. JSDL is XML-based language, and the extensible nature of XML allows for extending the base specification to include additional parameters in order to support the privacy framework presented in this paper. The decision was made to include the PPL value supplied by the consumer as an attribute of the JobDefinition element; since the PPL value modifies the job and does not change the job definition. The broker service collects the information and accepts the message for processing. On the resource provider side the resource provider uses a GIS (Grid Information System) to register itself with the Grid. In our case GIS is provided by DF. The resource matching component of the broker service uses DF to look up its capabilities with the broker service. The resource provider uses the registration service provided by the JADE platform to communicate its computing and data capabilities including the PPL value it can provide for job execution. The content of the registration message is a JAVA Properties based structure, key/value pairs, which represent the agents computing and data capabilities.

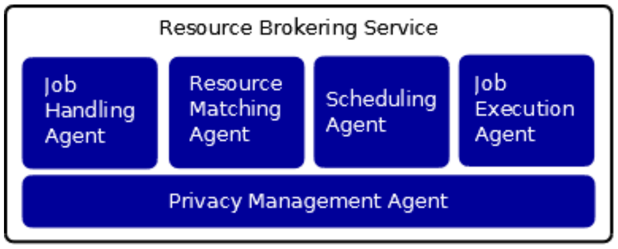

Figure 5-3: Resource-Broker with Privacy Management

The interaction between the different entities in the system is facilitated by a high-level protocol, ContractNET. ContractNET is a task-sharing protocol, and it can serve as the protocol for the base communication between the broker and the external entities. The protocol was extended to use an additional layer to support internal communication between the different agents within the broker service. This approach provides the required flexibility for the system to adapt at runtime. Internal to the broker, the communication component takes the received message from a resource consumer and puts that in a JSON format. The received content then becomes a value for the name job. The JSON structure adds flexibility in communication between internal component and the broker where each component can add additional name-value pairs to the structure and extract information from the structure which itself requires, leaving the rest of the structure unmodified for the next component. This is part of the information processed by the broker and creates new types of information, based on the information collection that accumulated from accepting the job and performing matching of the capabilities.

```
<JobDefinition ... PPL=”8”>
...
</JobDefinition >
```

Figure 5-4: Job definition and PPL value

5.3 Resource-Broker

The resource broker implementation is logically divided into two main parts. At the core is the agent technology that provides the implementation of functionality for each component of the resource broker, as well as the agent abstraction of the resource consumer and resource provider. Second, the interaction protocol used for communication and interaction between the different agents within the system.

The second part of the implementation is with respect to the particular functionality of each component as presented in the architecture. Each resource consumer and resource provider are abstracted using the agent technology, and interact with each other through the broker service. The abstraction here gives the ability to match resource consumers with available resource providers at the capability level, as well as allow for the broker service to appear as a black box to the grid participants. Here, we can treat the broker service as an information system which collects information from the resource consumers and resource providers, process the given information, and disseminate some information to the resource consumers and resource providers where appropriate.

5.3.1 Job Handling

The Job-Handling is responsible for receiving the initial request for the resource-consumer agent. A job-handling agent is the agent who responds to the CFP message to the "*broker*" service, where the resource-consumer looks up all available "*broker*" services on the Grid and sends a CFP message to initiate the job execution process. The supported job definition language is the JSDL, but this is where the support for other languages can be added such as RSL, etc. The job-handling agent can perform any job parsing and formatting needed before sending it off to the resource-matching agent. Job-handling agents interact with the resource-matching agent as part of the brokering process and job execution lifecycle.

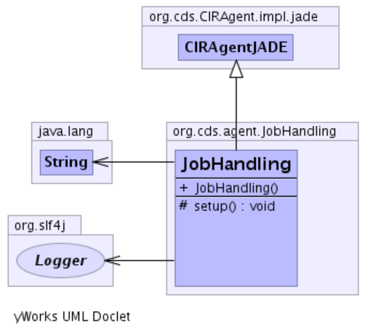

Figure 5-5: Job-Handling class diagram

5.3.2 Resource Matching

Resource-matching is responsible for matching the job requests to available resource providers. Resource-matching agent uses the GIS (**G**rid **I**nformation **S**ystem) in our case provided through the JADE platform by DF (**D**irectory **F**acilitator) to match the requested jobs with available computing and data resources. The resource-matching agent interacts with the job-handling agent where it receives the job definition. Upon receiving the job definition, the matching of resources to the job definition is performed, and further a required scheduling agent is selected to perform scheduling for the job. The proposed architecture allows for multiple scheduling agents to exist at runtime. The resource-matching agent selects the scheduling mechanism to be used based on the consumer request that is part of the job definition sent from resource-consumer agent.

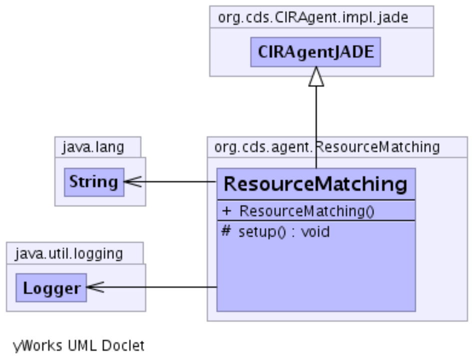

Figure 5-6: Resource-Matching class diagram

5.3.3 Scheduler

Simple-Scheduling is one of the scheduling components implemented for this project. Other scheduling mechanisms such as iterative market auction and one-shot auction have also been implemented. The scheduling agent receives the job definition coupled with a list of resources that are capable of performing the job execution based on the resource requirements in the job definition. The Simple-Scheduling agent performs the scheduling of the job execution by choosing a random number in the list of resources, provided by the matching step.

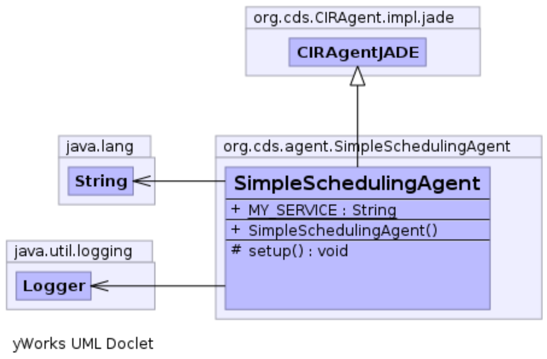

Figure 5-7: Simple-Scheduler class diagram

5.3.4 Job Execution

The role of the Job-Execution is to perform the job execution requests from the scheduling agent. The Job-Execution agent sends the scheduled job to the selected Resource-Provider at the scheduled time.

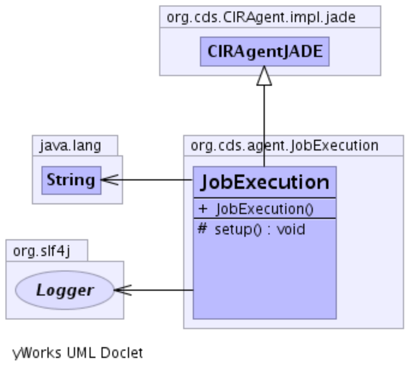

Figure 5-8: Job-Execution class diagram

5.4 Resource Provider

The resource-provider represents a computing or data resource participating in the grid. A resource can be a single machine making its computing and data resource available or it can be a cluster of machines. In the implementation, the Globus Toolkit has been utilized to perform the execution of the tasks at the resource end. The resource-provider agent uses GRAM (Globus Resource Allocation Manager) to submit jobs for execution at the local level. GRAM does the local resource allocation to perform job execution. The interaction between Globus and the resource-provider agent is done through java libraries provided by the Globus project. GramJob is a java class used by the resource-provider agent and provides the required functionality to execute, monitor and cancel job requests.

On the resource provider side, resource-provider agent abstracts the computing node resource as computing and data capabilities. The resource-provider agent utilizes the Globus Toolkit to gain access to the computing node resources for job execution. The integration with Globus Toolkit and ability to integrate through other middleware provides us with the advantage to manage the task execution lifecycle on different platforms. Providers' capabilities are registered in the Grid through the GIS in our case the DF.

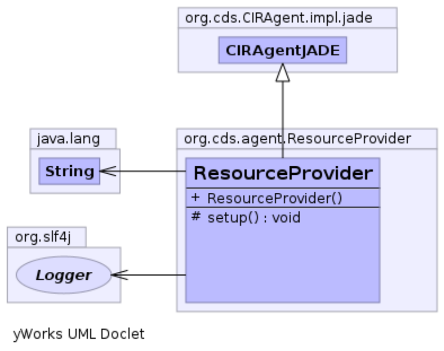

Figure 5-9: Resource-Provider class diagram

Although our implementation takes advantage of the JADE platform and its supporting agents, such as the directory facilitator (DF), agent management service (AMS), and agent communication center (ACC), the implementation and design of the application is based on the CIR-Agent[36] architectures. The CIR-Agent architecture maps very well to our agent based system architecture, where each agents' preferences and behaviors are captured in terms of knowledge and capabilities. The CIR-Agent architecture also adds two other layers for the agent interaction itself, the interaction devices and a communication layer. The interaction devices allow the separation of the interaction from the capabilities and knowledge of the agent and the communication layer is responsible for composing, sending and receiving messages. Each agent in our architecture uses the CIR-Agent framework implemented in Java. The design of each agent is described in terms of its knowledge and capabilities. The agent’s knowledge includes the agent’s self-model, goals, and local history of the world. The agent’s knowledge also includes its desires, commitments, and intentions as related to its goals. The architecture separates the matching functionalities of the broker to match the job definition to available providers into it’s own unit. This approach eliminates any providers that cannot provide enough PPL for the consumer, therefore reducing the feasible solution space for the scheduling component.

The flexibility of the resource broker architecture allows for multiple scheduling mechanisms to exist at runtime. The implementation uses agent technology to represent each component in the Grid environment. The agent technology provides the interaction required for such entities to coordinate and therefore allows for the negotiation of the scheduling mechanism used. The output of the matching component provides a reduced solution space for the input for the scheduling component where the scheduling component can be decided on at runtime, based on information supplied by the consumer, or internal broker problem solving capability which can determine the best scheduling algorithm given the particular job constraints or objective function.

The implementation is based on the CIR-Agent architecture. The implementation has been structured around two interfaces. On the one side the CIRAgent interface to the agent framework, in our case JADE, by implementing the CIRAgentJADE class, and on the other side the Action interface which works on the environment and is used by the agent platform to perform work. In addition, our project also depends on the Globus middleware. The Globus Toolkit has been utilized for the job execution portion of the project at each computing resource.

5.5 Resource-Consumer

The resource-consumer agent abstracts the consumers of the Grid. It captures users preferences and interacts with the resource-broker to execute jobs on consumer behalf. Resource-consumer agent extends CIRAgentJADE. CIRAgentJADE implements the CIRAgent interface and captures the JADE platform specific implementation. The implementation of the resource-consumer agent is simplified to the implementation of the action required for achieving its goal. The consumer's goal is to submit the job definition to the resource broker for execution.

The class diagram of a resource-consumer is shown in Figure 5-10. There are minimal external dependencies and all platform dependent code is captured in CIRAgentJADE class.

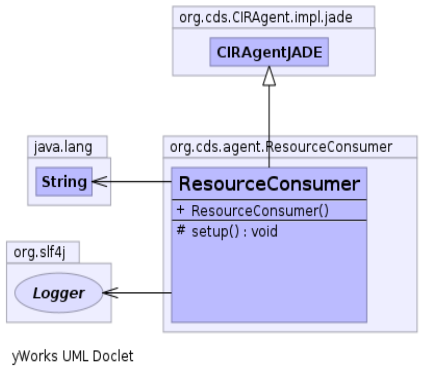

Figure 5-10. Resource-Consumer class diagram

5.6 Computing as a Platform: Prototype Implementation

A computing platform is an environment where a software program can execute. The implementation of the "Computing Platform" is composed of two parts. First part is the JADE platform that provides the ability for software agents to exist in the environment. JADE only provides a mechanism for communication and ability to build software agents. The solution extends the JADE platform to enable a new way to perform "Grid Computing". The "Grid Computing" platform as presented in the thesis is realized through the extension of the Grid Protocol Architecture, by providing a strict separation between the Application space and the lower levels of the "Grid Computing" capabilities.

The implementation of the prototype consists of two parts. First, software programs run on the platform and they are represented as software agents. Secondly, the flow of information has been verified through the logs attached in Appendix B – Experiment Logs. The scenarios performed show how during the job lifecycle there is no direct knowledge of the computing resources making up the grid environment. The computing has been transformed in term of abstracting the resources themselves and by extending the Grid Protocol Architecture to be more strict about the access to each layer of the protocol. Only the *Resource-Provider* is aware of the computing resources that provide the required capability to execute the job and physical access to the computing nodes.

5.6.1 Programming Languages Used

The core of the solution is implemented using the Java programming language. Other languages such as Bash scripting and Python are utilized in different parts of the solution. Bash scripting is used to start and stop each software artifact on the deployed host. The Python language is used for performance testing as well integration testing of each agent.

5.6.2 JADE Platform

JADE (JAVA Agent Development Framework) is an open source platform for an agent-based development of applications. JADE implements FIPA (Foundation for Intelligent Physical Agent) specifications for interoperable intelligent multi-agent systems. JADE is considered a platform because it enables the building of agent-based applications [62]. Agent-based architecture is a computing paradigm, which is known to be the next evolution of computing after OO (Object Oriented). Agent abstraction provides some key characteristics to the system, such as adaptability, autonomy, and interoperability. Agent-oriented is the next generation for software engineering paradigm, programming methodologies and computational paradigms.

JADE is a FIPA-compliant agent framework. The JADE platform provides communication facilities between agents using ACL. It also provides other agent facilities for agent discovery and communication with specialized agents such as AMS, DF and ACC. JADE is open source software and distributed by TiLab and the University of Parma.

The choice of technology for the implementation was to select and use JADE as the agent platform. JADE is the implementation of the FIPA (Foundation for Intelligent Physical Agent) specification for a multi-agent system. JADE is an agent platform that allows for the development of agent-based applications in compliance with FIPA; the platform provides the minimum running environment where agents exist and interact, as well as can be configured to run as a distributed platform with failover functionality. Other management facilities with JADE, such as DF (Directory Facilitator), AMS (Agent Management Service) and ACC (Agent Communication Center) are provided by specialized agents deployed at platform start-up. DF is a directory facilitator agent that provides yellow pages lookup services. AMS is an agent management system, which provides white pages and life-cycle service; it maintains a directory of agent IDs and their state. ACC is the software component orchestrating the exchange of messages between agents, across the distributed network of JADE nodes. All this provides an abstraction from the computing resources and each agent that runs on the platform is not directly tied to a particular computing resource.

5.7 Prototype Validation and Verification

In order to assure that the solution was done correctly, the verification of the proposed solution was done through automated unit tests and manual testing with console logs. The console logs were verify against requirements and expected execution of each scenario described bellow. Logs have been captured in Appendix B – Experiment Logs and are explained in the subsection bellow. In addition the yWorks tool was used to produce the class diagrams from code to verify the design, as described in Section 5.3, Section 5.4 and Section 5.5.

Prototype validation has been chosen as the tool to provide results from the implemented solution. Validating the prototype through a working scenario provides the required data to determine the feasibility and effectiveness of a new way to study Grid Computing. Two scenarios have been prepared to validate each part of the implemented model by looking at the information flow and interaction between the computing entities. In the controlled part of the experiment, the resource-broker has been omitted from the job submission flow. This required changes to the part of the system that deals with job submissions. The jobs submitted for execution invoked a native call through GRAM, a native interface part of the Globus Toolkit. The command line tool makes the job submission directly to the local resource manager of the Globus middleware. In the job submission process, the submitter that could be either an end user or a broker needs to specify the computing resource that will be responsible for performing the computation.

Here the resource allocation happens at the Application level as discussed in Chapter 3. Therefore, it shows that the network is used merely as a communication medium and the job arrives at the computing node for execution.

In the second scenario, the job is submitted to the resource broker. It includes computing resource capabilities required for job execution. The components of the resource broker responsible for different parts of the job execution provided the computing capability to allocate the necessary resources to execute the job. The computing in this scenario has not been performed at the specific computing node, and it was the responsibility of the Grid Platform to perform the computation that involved two parts, Computing Capability and Resource Allocation.

The system has been deployed on five physical machines as shown in Figure 5-11, with a sixth machine that extends the grid infrastructure through a private cloud implementation, not shown in the diagram. The details about each deployed node are captured in Table 5-1. Four old desktop machines were used to construct a Globus computing cluster. One of the newer desktops has been used as a grid portal, and then recently purchased dual-socket Xeon machine with 16 virtual cores were used to create a private cloud environment using OpenNebula software. The OpenNebula is an open source software system that enables data centre virtualization as well as cloud management. The cloud management module allows for a data centre to extend its infrastructure capabilities to other public cloud providers such as AWS or Azure. Extending a private cloud infrastructure allows an organization to support high peak workloads that stretch beyond the capacity limits of the private cloud infrastructure. OpenNebula was used in this environment to serve both of the features. It was used to virtualize the powerful 16-core machine where a number of on-demand virtual machine instances were launched to simulate different scenarios for the experiments; also, a connector was configured to the external public cloud on AWS.

Table 5-1: List of computing nodes

Name	**Specification**	**Notes**
cds-station1	Pentium 4/512MB/40GB	Grid Node
cds-	Pentium	Grid Node

station2	4/512MB/40GB	
cds-station3	Pentium 4/512MB/40GB	Grid Node
cds-station4	Pentium 4/512MB/40GB	Grid Node
cds-grid1	Core2 Duo/4GB/500GB	grid portal
cds-server	2 Xeon/24GB/1T	private cloud

The setup and configuration of each of the software technologies used in the environment for the experiments have been outlined in the following subsections of this chapter. The four desktop machines named cds-station1 through four have been configured with running Globus Toolkit configured as an independent Globus computing cluster, with multiple physical computing nodes, as described in Section 5.7.1. In Section 5.7.2 the details about configuration for JADE platform is captured. Section 5.7.4 depicts the deployment and configuration of the resource-broker service itself and its dependencies.

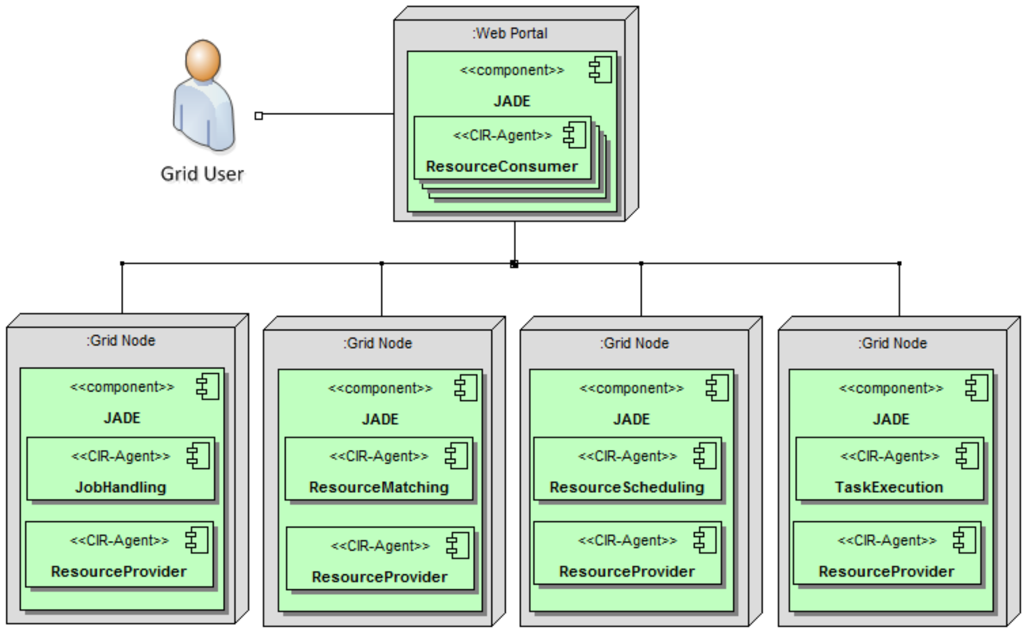

Figure 5-11. Deployment diagram

5.7.1 Globus Toolkit

Globus® Toolkit is an open source fundamental enabling technology for building grid-computing infrastructure [37]. Globus Toolkit is often referred to as middleware, where it provides access and is a layer of abstraction to the collection of heterogeneous and geographically distributed computing resources. The project developed as part of the work on the thesis uses version 5.0.4 of Globus. A collection of machines has been selected and configured to be part of the Globus cluster to perform computing jobs for the experiment. Globus has been configured with a self-signed certificate that enables a secure connection between the computing nodes. It also creates a trusted environment. Globus provides the ability to create a certificate authority. The certificate authority is used to generate a security certificate. Each computing node participating in the grid-computing environment uses a security certificate generated for that computing node. Due to the nature of the technology used in providing the secure communication in Globus, and the signing of the security certificates, the generated certificate is specific to each computing node. The command line tools supplied and distributed with Globus Toolkit also allows for creating and processing of certificate requests to generate a valid security certificate. Each certificate has an expiry date and can be revoked at any time by the middleware if access needs to be restricted.

The signed certificates are created as part of the Globus installation steps, and all tests have been performed using the secure environment. In addition, each machine making up the computing grid has been secured using IPTables firewall and configured to only allow specific known ports from a specific IP addresses to be open and allow for a host to host communication.

5.7.2 JADE

The configuration for the experiment has been set up in such a way it resamples a real world realistic setup. JADE platform has been configured and deployed with multiple main-containers. This type of configuration uses the Main Container Replication Service to create redundancy and failover. Such a configuration creates high availability type setup, where if any one or two hosts fail, the platform will still function and agents running on the platform will be moved to the hosts that are still in operations. The agents running on the platform are not aware of this setup, nor they are aware of the host they are running on. There is no direct dependency of the physical computing host and agents that exist and communicate on the JADE platform.

5.7.3 Entity Structure

The entity structure validation was done through examining the identified open issues as discussed in Chapter 3. In the context of the “computing entity” there is one key open issue that applies to the “computing entity” and can be used to validate the entity structure for the new way of doing grid-computing. The structure of the "computing entity" shall give a level of autonomy. Autonomy defines the ability to self-govern and interact with other entities in the system through coordination. In the scenario, as described in the previous section, the interaction between entities is shown to be done at the capability level and not through access and control. Access and control approach is part of the current Grid Protocol Architecture that provides protocols to access individual computing resources. Capability level interaction allows for a more efficient approach to resource allocation with gives the "computing entities" full control over their availability and capacity to perform the computation.

5.7.4 Resource-Broker

The resource-broker service is divided into four key components, each compiled into a separate JAR executable. Each component implemented as a software agent from the resource-broker service runs inside the main-container, one agent per main-container configuration. A container in this context is an instance of JADE main service running in a single JVM. The system has been deployed with four main-containers in a failover configuration as described in the JADE section above. Each main-container runs on a separate physical machine in order to properly simulate real world environment where all communication happens over the Internet and not within a single machine.

Resource-provider agents have been deployed in peripheral JADE container where the main-container configuration endpoint is specified at the time of execution. The main-container address provided is a fully qualified domain name or FQDN for short, although the peripheral and main-container run on the same physical machine, due to the limited physical machine resources in our lab, this is a well documented deployment model. This forces the deployed configuration to use the external endpoints during agent communication and therefore can be easily migrated to different physical machines. The resource-provider is deployed on the computing node that it is representing, however the configuration of the resource-provider specifies the FQDN as well and therefore again forces the communication to occur using the publically accessible end-points. This configuration allows the JADE platform to run on a set of dedicated machines that would be separate from the available computing resource making up the grid infrastructure.

Resource-consumer agents are deployed on a separate JADE container and connected to the JADE main-container over the Internet. A resource-consumer agent is created for each user in the system. The jobs are submitted to the resource-consumer through the file system, with a plan to make this available as an API. A specific directory on the local files system is monitored for change. At the time directory change is detected, the most recent file is read into memory and parsed into a *java.lang.String* object. The *java.lang.String* representation of the job definition is then added to the content of the ACL message as part of the CFP message to the resource broker service.

Running the experiment produced the following results where the process of job submission, matching, scheduling, and execution is captured in the process log messages and attached in Appendix B – Experiment Logs. The attached content of the files show the logical steps each of the components of the Resource-Broker is performing as the submitted job goes through its life-cycle. The captured logs are described in the subsection below for each of the components of the implemented solution.

5.7.5 Resource-Consumer Agent

The resource-consumer agent is configured to read a job definition file from a local file system for the purpose of this simulation. The module responsible for reading the job definition can be easily swapped with a REST API interface as it implements an Action interface and is fully autonomous. Once job definition is loaded and parsed, the agent proceeds to the next state of execution and submits the job to the resource-broker for processing.

During this process, the resource-consumer is not aware of the resource provider that carries the execution. The interaction between the resource-consumer and resource-broker is done at the capability level abstracting all resources connected to the grid. In addition resource-consumer is not aware of any of the components/agents that make up the resource-broker service. The communication is done at the resource-broker interface.

5.7.6 Job-Handling Agent

The job-handling agent is the component of the resource-broker that handles job when first submitted. It is the agent that listens for the "rbroker" topic and handles the initial jobs submission from the resource-consumer agent. This is based on the implementation that the communication between agents is done through message routing based on specific topic and not direct communication between resource IP address and port. Due to the fact that the state of the communication topic "rbroker", the job-handling agent picks up the received message and sets its internal state to accomplish its goal of processing this state of the job lifecycle. The job-handling agent is only responsible for part of the job lifecycle and therefore once it reaches its internal goal of accepting and validating the submitted job, it delegates other agents' to take on the next state of the job lifecycle that is beyond its capabilities. The next state of the job lifecycle is called "matching.

5.7.7 Resource-Matching Agent

The resource-matching agent is the agent that receives the job specification when the job is in "matching" state. The resource-matching agent has two responsibilities; the first is to perform matching of the resource definition, followed by matching of the scheduling agent. Supporting multiple scheduling agents provides the flexibility of the solution to have multiple scheduling techniques that are available and selectable at run-time.

5.7.8 Scheduling Agent(s)

The scheduling agent or agents, where multiple scheduling techniques are supported by resource-broker ecosystem. Each scheduling agent implements a different scheduling protocol. The job specification language allows extending the job definition with additional fields that enable the system to adapt itself at runtime to the different capabilities provided by the resource-broker. The prototype has used the SimpleSchedulingAgent in this scenario, where the scheduling algorithm is based on a first available resource in the list that is capable of performing the job based on the job specifications and resource capabilities.

5.7.9 Job Execution Agent

The job-execution agent is responsible for performing the “execution” of the job. The “execution” is a state in the job lifecycle that uses the computing node to perform computation. The job-execution agent registers this capability with the resource-broker platform and therefore receives job definitions for all jobs that read this state. The performance can be improved by deploying multiple instances of the same agent, however, this is not the focus of this thesis. The “execution”a state of the job lifecycle performs the *Resource Allocation*, part of the computing platform. This is the only part of the system that has the ability to communicate with resource-providers and submit the requested job for execution to the specific computing resource. The communication with each capable resource-provider is still done at the platform level without the direct knowledge of the resource endpoints or network address. Once the commitment is reached between the selected resource-provider and the resource-broker, the job specification is sent to the selected resource-provider for execution. The job now moves to the “executing” state.

5.7.10 Resource Provider Agent

The resource-provider agent is an agent representation of the physical computing resource participating in the Grid. The resource-provider agent has the ability to execute the job on that physical resource or a cluster/pool of resources. The resource-provider agent is not part of the resource-broker, but it is an agent representation of the computing resources that make up the grid-computing infrastructure. This type of representation allows for a consistent design and simplifies the integration with physical computing infrastructure.

5.8 Implementation Challenges

There were a number of challenges in implementing the proposed solution and performing the experiment. The implementation has to take into account that the solution cannot be a single executable service and has to be distributed. Also each distributed component shall not be aware of direct endpoints of other components that it needs to communicate. The communication and resource allocation during the job workflow should be done without any direct awareness of computing resources that make up the grid-computing infrastructure.

5.8.1 Architecture

The proposed architecture requires to implement different components for each logical function of the resource broker and job workflow. The implemented solution could not be deployed as a single monolithic executable or service running on a single computing resource, as it would invalidate the experimental validation of the thesis. The solution has been implemented into a minimum of four different, and separate executable components that can run as a service with the ability to communicate through the network. The deployed components could not have direct knowledge of each other physical computing resource. Using JADE as the computing platform, where each of the functional components has been abstracted using agent technology solved this challenge. This allowed for agent-to-agent communication at a higher level, where the network was simply used for exchange of message, without direct knowledge of physical endpoints tied to computing infrastructure.

5.9 Dynamic Selection of Scheduler

During the implementation of the proposed solution it was evident that with minimal changes, the proposed solution could provide a more flexible approach to scheduling. In the reviewed literature, current implementations provide a specific type of scheduling algorithm that was selected at design time. Such an approach is not adequate for the next generation of computing needs.

Another experiment was constructed to show the flexibility of the design and implementation of the proposed solution. Resource broker uses agent-based architecture, where the key features of the resource-broker have been split up and wrapped with agent technology to provide autonomy, interoperability, and capability-based interdependency during run-time. This approach in the implementation allows the agents to work together to perform the objectives of the resource broker as a whole. The agent uses a communication protocol in order to coordinate their tasks in order to accomplish the specific goal of job matching and job allocation. The communication protocol provides the means for communication and exchange of messages, where the problem solver component, allows for the logic related to accomplishing the specific goal of the agent.

In the case of the Scheduling Agent, the interaction of the Scheduling Agent and the Job resource-matching agent allows for a dynamic selection of the Scheduling Agent type, where a number of Scheduling Agent types can exist in the system at any point in time, and coordination of the agents allows for the dynamic selection of the agent which is best suited to perform the scheduling of the submitted job. The coordination of the different Scheduling Agent types is done through job definition, where a specific scheduling algorithm can be explicitly specified. This approach can be further extended with a deterministic approach to determine the scheduling algorithm required. All this happens during the job handling and matching process, which means the communication of the different software artifacts are not known at design time, and only at run-time.

In the current implementation of the algorithm that is responsible for the negotiation and selection of the Scheduling agent has been kept very simple; for the purpose of the thesis and to provide the proof of concept solution. The implementation can be further extended with additional logic as captured in Section 6.2. The selection algorithm is a simple string matching, where the Scheduling Agent is selected by explicitly specifying the name of the agent in the job definition structure. The chosen language for the job definition is JSDL. The specification of the language allows for extending the definition through the use of custom fields. The additional fields are added to the job definition, and as the job is submitted to the resource-broker and passed among the different components of the resource-broker service, each component extracts the parts of the job definition related to it, where only the specific component understands the custom fields.

5.10 Summary

In the implementation of the proposed solution, it was not trivial to ensure that the proposed architecture will be clearly preserved. The distributed deployment approach of the software components simplified the experimental validation. The experimental validation was used to validate the thesis and show how the challenges in the implementation of the solution have been dealt with. The validation was done through two different scenarios that illustrated how computing is done today on existing grid infrastructure and how the computing should evolve to be true grid computing.

The added flexibility of the design also allowed the dynamic selection of the scheduler at run-time, where scheduler was selected based on the specification of the submitted job.

Chapter 6

6 Summary and Conclusions

The thesis explored the concept of cooperative distributed systems and applied it to the grid computing environment. It is focused on the functionality of resource brokering aspect of grid computing. The objective was to view the grid as an open environment and apply open environment characteristics to the grid-computing paradigm. Open environment has a dynamically changing architecture where participants of the grid have the ability to join and leave at will, yet are able to participate and coordinate their execution with other participants of the environment. The proposed architecture for resource brokering allows for a greater flexibility in all aspects of, job handling, scheduling and monitoring. Additional support for different grid middleware solutions can be added, and is further discussed in Section 6.2.

The core contribution is the extension of the Grid Protocol Architecture. The thesis does a deep dive into the protocol architecture and analysis that have put Grid Computing on a specific path, diverging from the grand vision of what grid computing aught to be. The extension to the Grid Protocol Architecture allows for a new direction that can help bring computing on the Grid closer to the vision of utility computing. The thesis showns how the computing inside a single machine, where the computing resources are CPU, memory and disk storage are the means and provide the required computational power to perform any form of computation and extends the fundamental computing concepts to the network. In traditional desktop computers the operating system layer exists as a computing layer on top of the physical resources that provide the means to perform the computation. In such view it is clearly visible that the computing happens at the operating system level, not at the resources themselves; “Network is the computer”, was the phrase put forward by John Gage of Sun Microsystems a few decades ago. This analogy is extended here, where the mapping between the single node resources and resource forming a grid; where now the pool of geographically distributed computing nodes, connected in a grid-like fashion provide the means for computation to be performed, yet the computation itself happens at the grid level, as shown in Figure 4-2.

6.1 Summary of Contributions

The main goal of the thesis was to extend grid-computing environment to support open environment. An open environment extends the Grid with a set of characteristics that make it current; today, with the evolution of the computing platforms, and introduction of IoT (Internet of Things) platforms, there is an urgent need for computing entities to have the ability to join and leave the platform as needed, to interact and cooperate.

In the design of the resource-broker, the fundamental requirement is the perspective of open environment and its characteristics have been successfully applied and it has been shown how entities have the ability to join and leave the environment at runtime without affecting job execution. The architecture of the resource broker enabled autonomy of the computing entities as well as the entities making up the resource broker itself. The issues with the flexibility of the design and interoperability have been addressed, with the main contribution of flexibility for the scheduling algorithm that can be selected at run-time. This carries advantages over existing resource broker designs, where the scheduling algorithm is part of the core implementation of the resource broker. Such an approach makes the resource broker implementation very specific to the application domain, and the architecture very rigid. The outlined design the scheduling algorithm can be chosen at run-time based on the job specification and objective function. The supplied parameters in the submitted job definition are used to correctly pick and inject the correct scheduler with the required functionality at run-time, where the resource broker has the ability to make the appropriate selection when scheduling the job for execution.

In collaborative work with the CDS group, scheduling with privacy concerns model has been implemented by simply extending the proposed architecture. A co-authored paper on that topic has been published, where the analysis and development of the scheduling model that takes privacy concerns of entities has been expanded to the grid, with architecture that enables privacy concerns in the scheduling decisions. The approach taken in the design of the solution also helped reduce the complexity of the overall problem; by splitting the problem into several parts. Applying brokering architecture, adding autonomy to the resource consumers and resource producers, and applying open system characteristics to grid computing.

The flexibility of the solution also allows extending the architecture and adopting the solution for privacy. Working together with a colleague in the CDS-Group, we proposed a privacy framework for open environments by extending the proposed solution with a privacy-broker. The solution enabled the cooperation of computing resources under the desired level of privacy protection. Privacy Protection Level also known as PPL is the contribution put forward by my colleague, Samani, A., who is also the main author of the paper. My contribution was focused on proposing the architecture for the solution in the context of open environment. The work was published in conference proceedings at the 2013 PASSAT[57]. The proposed privacy model was able to reduce the risk of privacy violations in entity interactions. The design of the resource-broker as proposed in this thesis was extended to support the privacy model proposed in the co-authored paper and applied to grid computing environment. Such implementation showed the flexibility of the proposed architectural solution as described in this thesis. With a simple extension of the proposed architecture, the resource broker was capable of brokering requests in the context of privacy and the exchange of information at all three levels: information collection, information processing and information dissemination. The goal of the collaborative work was to use the provided and formal treatment of "privacy" as a fundamental computational concept in CDS paradigm and implement the privacy-aware CDS framework as a CDS platform with the ability to support interaction-based privacy protection[57]. This work also allowed for validation of the proposed solution, and its extensibility to support other computational frameworks.

In addition, the work here was directed to provide a fragment of work done under the CDS group, where capability-based coordination in cooperative distributed systems is the foundation of resource brokering as applied to grid computing. An agent-based approach has been implemented for the solution, to enable brokering of capability-based resources in a grid environment. All grid resources have been modeled as CIR-Agent, and with this approach, a flexible solution was implemented which enabled seamless capability-based brokering to all the participants within CDS. The proposed architecture minimized the complexity encountered in the direct interaction architectures.

An important class of distributed systems is CDS, in which entities are able to exercise some degree of authority in sharing their capabilities. Entities in this paradigm are expected to cooperate to achieve individual or collective goals. Due to interdependency problem among entities, they require the coordination of their activities using interactions. In the message-based form of interactions, entities exchange information through autonomous and self-interested entities, and thus their privacy becomes a concern. In CDS, solutions are accomplished through the participation of several entities where each has only part of the solution. This positions CDS as a computation platform in which the computation occurs at entities' interactions. This entails that privacy challenges in CDS are the concerns associated with the computation happening at the interaction level [46].

6.2 Future Work

One of the areas to expand the research is by evaluating solutions for a distributed resource broker architecture, where a number of resource broker services are able to coordinate in real time and provide access to their grid-connected resources. Based on the research provided here, a solution that encompasses autonomy combined with peer-to-peer or neural network properties would be a candidate for a good solution. Such architecture shall apply to the brokering layer, enabling a cooperative distributed system solution to a network of resource brokers.

Along similar lines another option would be to revisit computing as utility [7], where the current form of cloud computing is an attempt to bring computing as utility to the masses, however, it is not there yet. A user still needs to understand the needs and required computing power for which it has to provision the cloud in order to effectively run services. Resource brokering will play a key role here, where the evolution of cloud computing towards true utility computing will form. Although the cloud-computing paradigm has pushed forward a new approach towards service reliability and scalability with on-demand infrastructure provisioning, it has further room to grow to be classified as the true equivalent of utility as computing. Currently development and operations combined still require manual or partially automated ways of scaling services and infrastructure. In my view, the resource brokering will play a key role in automating and providing the computing needs to the autonomous service running anywhere on any device, that will have the ability to draw on demand computing needs, in a similar way that dump devices currently draw power from power companies.

Further work related to the flexibility of the scheduling can be extended as well. The extension of the current work could focus on the ability to best match for the scheduling algorithm based on the job specification or the objective function. In the current implementation, the selection is done through simple matching of the requested scheduling algorithm as part of the job specification. However, in a real deployment, the scheduling algorithm may be best selected by the other or multiple means. The system can be configured to find best optimum-scheduling algorithm based on the supplied parameters or objective function, as it currently does simple matching.

References

[1] Abramson, D., Buyya, R., and Giddy, J. (2002). Future Generation Computer Systems, 18(8), 1061-1074. doi:10.1016/S0167-739X(02)00085-7

[2] Abramson, D., Giddy, J. and Kotler, L. (2000). "High performance parametric modeling with Nimrod/G: killer application for the global grid?", pp. 520.

[3] Abramson, D., Sosic, R., Foster, I., Giddy, J., Lewis, A. and White, N. (1996). The Nimrod computational workbench: A case study in desktop metacomputing (No. ANL/MCS-P--594-0596; CONF-961104--9). Argonne National Lab., IL (United States).

[4] Abramson, D., Sosic, R., Giddy, J. and Hall, B. (1995). August. Nimrod: a tool for performing parametrised simulations using distributed workstations. In High Performance Distributed Computing, 1995., Proceedings of the Fourth IEEE International Symposium on (pp. 112-121). IEEE.

[5] Aburukba, R.O. (2013). *Decentralized resource scheduling in Grid/Cloud computing*, School of Graduate and Postdoctoral Studies, University of Western Ontario.

[6] Allcock W., Bester J., Bresnahan, J., Chervenak, A., Liming, L., Meder, S., and S. Tuecke, S. (2002). "GridFTP protocol specification", *Technical report, Global Grid Forum GridFTP Working Group*, September 2002.

[7] Andrzejak, A., Reinefeld, A., Schintke, F. & Schütt, T. (2006). "On Adaptability in Grid Systems" in Springer US, Boston, MA, pp. 29-46.

[8] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. & Zaharia, M. (2010). *A view of cloud computing*, ACM, New York.

[9] Berman, F., Fox, G.C. & Hey, A.J.G. (2003). "Grid computing: making the global infrastructure a reality", J. Wiley, Hoboken, NJ;Chichester, England;.

[10] Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S., Hayes, J., Obertelli, G., Schopf, J., Shao, G., Smallen, S., Spring, N., Su, A. & Zagorodnov, D. (2003). "Adaptive computing on the Grid using AppLeS", *IEEE Transactions on Parallel and Distributed Systems,* vol. 14, no. 4, pp. 369-382.

[11] Berman, F., Wolski, R., Figueira, S., Schopf, J. and Shao, G., (1996). Application-level scheduling on distributed heterogeneous networks. In Supercomputing, 1996. Proceedings of the 1996 ACM/IEEE Conference on (pp. 39-39). IEEE.

[12] Buyya, R., Abramson, D. & Giddy, J. (2000). "Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid", pp. 283.

[13] Buyya, R., Abramson, D. and Giddy, J., (2000). June. An Economy Driven Resource Management Architecture for Global Computational Power Grids. In PDPTA (pp. 26-29).

[14] Buyya, R., Stockinger, H., Giddy, J. & Abramson, D. (2001). "Economic models for management of resources in peer-to-peer and grid computing", .

[15] Cao, J., Jarvis, S.A., Saini, S., Kerbyson, D.J. & Nudd, G.R. (2002). "ARMS: An Agent-Based Resource Management System for Grid Computing", *Scientific Programming,* vol. 10, no. 2, pp. 135-148.

[16] Ch, V., Laxmi, L. & Somasundaram, K. (2014). "Application Level Scheduling (APPLeS) in Grid with Quality of Service (QoS)", *International Journal of Grid Computing & Applications,* vol. 5, no. 2, pp. 1-10.

[17] Chapin, S.J., Katramatos, D., Karpovich, J. & Grimshaw, A. (1999). "Resource management in Legion", *Future Generation Computer Systems,* vol. 15, no. 5, pp. 583-594.

[18] Clarens (2007). visited 6 July 2017, http://clarens.sourceforge.net/

[19] Clouds Lab (2017). visited 6 July 2017, http://cloudbus.org/678/Chap17.pdf

[20] Czajkowski, K., Foster, I. and Kesselman, C., (1999). Resource co-allocation in computational grids. In High Performance Distributed Computing, 1999. Proceedings. The Eighth International Symposium on (pp. 219-228). IEEE.

[21] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. & Tuecke, S. (1998). "A resource management architecture for metacomputing systems", , pp. 62.

[22] Czajkowski, K., Foster, I., Kesselman, C., Sander, V. & Tuecke, S. (2002). "SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems", , pp. 153.

[23] Dumitrescu, C., Raicu, I. & Foster, I. (2005). "DI-GRUBER: A Distributed Approach to Grid Resource Brokering", IEEE Computer Society, , pp. 38.

[24] Dumitrescu, C.L., Raicu, I. & Foster, I. (2007). "The Design, Usage, and Performance of GRUBER: A Grid Usage Service Level Agreement based BrokERing Infrastructure", *Journal of Grid Computing,* vol. 5, no. 1, pp. 99-126.

[25] Eerola, P., Konya, B., Smirnova, O., Ekelof, T., Ellert, M., Hansen, J.R., Nielsen, J.L., Waananen, A., Konstantinov, A., Herrala, J., Tuisku, M., Myklebust, T., Ould-Saada, F., Vinter, B. (2003). "The Nordugrid production grid infrastructure, status and plans", *Proceedings. First Latin American Web Congress*, , pp. 158-165.

[26] Ellert, M., at el. (2007). "Advanced Resource Connector middleware for lightweight computational Grids", *Future Generation Computer Systems,* vol. 23, no. 2, pp. 219-240.

[27] Elmroth, E. & Tordsson, J. (2006). "A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection" in Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1061-1070.

[28] Ferguson, D.F., Nikolaou, C., Sairamesh, J. and Yemini, Y., (1996). Economic models for allocating resources in computer systems. Market-based control: a paradigm for distributed resource allocation, pp.156-183.

[29] Forster I. (2002). “What is the Grid? A three point checklist”, http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf

[30] Foster I., Kesselman C., “Chapter 1: THE GRID IN A NUTSHELL”

[31] Foster, I. and Kesselman, C., (1997). Globus: A metacomputing infrastructure toolkit. The International Journal of Supercomputer Applications and High Performance Computing, 11(2), pp.115-128.

[32] Foster, I., Fidler, M., Roy, A., Sander, V. & Winkler, L. (2004). "End-to-end quality of service for high-end applications", *Computer Communications,* vol. 27, no. 14, pp. 1375-1388.

[33] Foster, I., Kesselman, C. & Tuecke, S. (2001). "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *The International Journal of High Performance Computing Applications,* vol. 15, no. 3, pp. 200-222.

[34] Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S., (2002). Grid services for distributed system integration. Computer, 35(6), pp.37-46.

[35] Frey, J., Tannenbaum, T., Livny, M., Foster, I. & Tuecke, S. (2001). "Condor-G: a computation management agent for multi-institutional grids", IEEE, , pp. 55.

[36] Ghenniwa, H.H. (1996). *Coordination in cooperative distributed systems*, ProQuest Dissertations Publishing.

[37] Globus (2017). visited 6 July 2017, http://toolkit.globus.org/toolkit/about.html

[38] Humphrey, M., Wasson, G., Jackson, K., Boverhof, J., Rodriguez, M., Gawor, J., Bester, J., Lang, S., Foster, I., Meder, S., Pickles, S. & Mc Keown, M. (2005). "State and events for Web services: a comparison of five WS-resource framework and WS-notification implementations", IEEE, , pp. 3.

[39] Imamagic, E., Radic, B. & Dobrenic, D. (2006). "An Approach to Grid Scheduling by Using Condor-G Matchmaking Mechanism", *Journal of Computing and Information Technology,* vol. 14, no. 4, pp. 329.

[40] ISO/IEC 7498-2: (1989).

[41] J. Brooke and D. Fellows. Draft discussion document for GPA-WG – Abstraction of functions for resource brokers. http://grid.lbl.gov/GPA/GGF7 rbdraft.pdf.

[42] Kumar N., Singh R., Arora V., Rohilla V. (2012). “Grid Computing”, I*nternational Journal of Advanced Research in Computer Science and Electronics Engineering*, vol. 1 no. 6, pp. 99-103.

[43] Kurowski, K., Nabrzyski, J. & Pukacki, J. (2001). "User preference driven multiobjective resource management in grid environments", IEEE, pp. 114.

[44] Lee, M., In, J. & Choi, E. (2006). "A scheduling middleware for data intensive applications on a grid", , pp. 1058.

[45] Malone, T. & Crowston, K. (1994). "The interdisciplinary study of coordination", *ACM Computing Surveys (CSUR),* vol. 26, no. 1, pp. 87-119.

[46] Masaud Wahaishi, A.M. (2007). *Brokering services for cooperative distributed systems: an agent privacy-based architecture*, Faculty of Graduate Studies, University of Western Ontario.

[47] Montagnat, J., Frohner, Á., Jouvenot, D., Pera, C., Kunszt, P., Koblitz, B., Santos, N., Loomis, C., Texier, R., Lingrand, D., Guio, P., Brito Da Rocha, R., Sobreira de Almeida, A. & Farkas, Z. (2008). "A Secure Grid Medical Data Manager Interfaced to the gLite Middleware", *Journal of Grid Computing,* vol. 6, no. 1, pp. 45-59.

[48] Nagariya, Sonal and Mishra, Mahindra, (2013). "Resource Scheduling in Grid Computing: A Survey", *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 10, pp. 735-739.

[49] NorduGrid (2017). Visited 6 July 2017, http://www.nordugrid.org

[50] OpenLDAP (2017). Visited 6 July 2017, http://www.openldap.org

[51] OpenSSL (2017). Visited 6 July 2017, http://www.openssl.org

[52] Plaszczak, P. & Wellner, R. (2006). *Grid computing: the savvy manager's guide,* Elsevier/Morgan Kaufmann, Amsterdam;Boston;.

[53] Raman, R., Livny, M. & Solomon, M. (1998). "Matchmaking: distributed resource management for high throughput computing", , pp. 140.

[54] Rani P. (2013). "Middleware and Toolkits in Grid Computing", *International Journal of Computer Applications*, vol. 65

[55] Resource Specification Language RSL v1.0., visited 6 July 2017, http://toolkit.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html

[56] Romanian GM. (2011). "Grid Computing Technology", *Database Systems Journal,* pp.13-22.

[57] Samani, A., Bienkowski, A.T., Aburukba, R. & Ghenniwa, H.H. (2013). "Privacy Framework for Open Environments", *IEEE*, pp. 460.

[58] Schwiegelshohn, U., at el. (2010). "Perspectives on grid computing", *Future Generation Computer Systems,* vol. 26, no. 8, pp. 1104-1115.

[59] Shen, W., Li, Y., Ghenniwa, H. and Wang, C., (2002). "Adaptive negotiation for agent-based grid computing", *Journal of the American Statistical Association,* vol. 97, no. 457, pp. 210-214.

[60] Swedish National Infrastructure for Computing. (2017). visited 6 July 2017, http://www.snic.vr.se/projects/swegrid

[61] Tedre, M. & Moisseinen, N. (2014). "Experiments in Computing: A Survey", The Scientific World Journal, vol. 2014, pp. 1-11.

[62] Telecom Italia (2017). Visited 6 July 2017, http://jade.tilab.com/

[63] Tomás, L., Caminero, A.C., Rana, O., Carrión, C. & Caminero, B. (2012). "A GridWay-based autonomic network-aware metascheduler", *Future Generation Computer Systems,* vol. 28, no. 7, pp. 1058-1069.

[64] Weiser, M. (2002). "The computer for the 21st Century", *IEEE Pervasive Computing,* vol. 1, no. 1, pp. 19-25.

[65] Wikipedia (2013). Visited 6 July 2017, https://en.wikipedia.org/wiki/Contract_Net_Protocol

[66] Wikipedia (2017). Sphinx, viewed 06 July 2017, https://en.wikipedia.org/wiki/Sphinx

[67] Wikipedia (2017). Visited 6 July 2017, https://en.wikipedia.org/wiki/Interdependence

[68] Wieringa, R. (2014). "Empirical research methods for technology validation: Scaling up to practice", Journal of systems and software, vol. 95, pp. 19.

[69] WISC (2017). Visited 6 July 2017, http://research.cs.wisc.edu/htcondor/

[70] Wolski, R., Plank, J.S., Brevik, J. & Bryan, T. (2001). "Analyzing Market-Based Resource Allocation Strategies for the Computational Grid", *The International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 258-281.

[71] Wooldridge, M., Jennings, N. & Kinny, D. (1999). "A methodology for agent-oriented analysis and design", *ACM*, , pp. 69.

[72] Xu, F., Pan, J. & Lu, W. (2009). "A Trust-Based Approach to Estimating the Confidence of the Software System in Open Environments", *Journal of Computer Science and Technology,* vol. 24, no. 2, pp. 373-385.

[73] Yousif, A., et al., (2015). “Job Scheduling Algorithms on Grid Computing: State-of-the Art", *International Journal of Grid and Distributed Computing,* vol. 8, no. 6, pp. 125-140.

[74] Zelkowitz, M.V. & Wallace, D.R. (1998). "Experimental Models for Validating Technology", Computer, vol. 31, no. 5, pp. 23-31.

Appendix A - yWorks

UML diagrams that represent the implementation classes for the software solution described in Chapter 5, have been automatically generated from the source code using community edition of the yWorks diagramming tool.

For more information regarding the diagramming technology please visit the following link, https://www.yworks.com/products/ydoc.

Appendix B – Experiment Logs

```
------ ResourceConsumer log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> SubmitJobAction
INFO ProblemSolverImpl.solve InitialStateAction -> SubmitJobAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> SubmitJobAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptJobActionLocalFile isPost: false
INFO ProblemSolverImpl.solve           action: AppendAgentNameAction isPost: true
INFO ProblemSolverImpl.solve           action: SubmitJobAction isPost: false
INFO ProblemSolverImpl.solve           action: CancelJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AppendAgentNameAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptJobActionLocalFile isPost: true
INFO ProblemSolverImpl.solve           action: AppendAgentNameAction isPost: false
INFO ProblemSolverImpl.solve           action: SubmitJobAction isPost: false
INFO ProblemSolverImpl.solve           action: CancelJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptJobActionLocalFile
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptJobActionLocalFile isPost: false
INFO ProblemSolverImpl.solve           action: AppendAgentNameAction isPost: false
INFO ProblemSolverImpl.solve           action: SubmitJobAction isPost: false
INFO ProblemSolverImpl.solve           action: CancelJobAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 3 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AcceptJobActionLocalFile
INFO CIRAgentJADE.execute       AppendAgentNameAction
INFO CIRAgentJADE.execute       SubmitJobAction
INFO AcceptJobActionLocalFile.action checking new files for cdsuser1
INFO AcceptJobActionLocalFile.action    found file: hostname.job
INFO AppendAgentNameAction.action adding consumer info
INFO SubmitJobAction.action submitting job for topic: rbroker
INFO CIRAgentJADE.sendMsg type: ACLMessage.REQUEST topic: rbroker

------ JobHandling.log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> SubmitMsgAction
INFO ProblemSolverImpl.solve InitialStateAction -> SubmitMsgAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> SubmitMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: true
INFO ProblemSolverImpl.solve           action: SubmitMsgAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: SubmitMsgAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 2 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AppendMsgAction
INFO CIRAgentJADE.execute       SubmitMsgAction
INFO AcceptMsgAction.action received null; blocking
INFO AcceptMsgAction.action received msg topic: rbroker
INFO AcceptMsgAction.action     msg validation: ok
INFO SubmitMsgAction.action submitting job for topic: matching
INFO CIRAgentJADE.sendMsg type: ACLMessage.REQUEST topic: matching
```

```
------ ResourceMatching.log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> ScheduleJobAction
INFO ProblemSolverImpl.solve InitialStateAction -> ScheduleJobAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> ScheduleJobAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchResourcesAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchSchedulerAction isPost: true
INFO ProblemSolverImpl.solve           action: ScheduleJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> MatchSchedulerAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchResourcesAction isPost: true
INFO ProblemSolverImpl.solve           action: MatchSchedulerAction isPost: false
INFO ProblemSolverImpl.solve           action: ScheduleJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> MatchResourcesAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: true
INFO ProblemSolverImpl.solve           action: MatchResourcesAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchSchedulerAction isPost: false
INFO ProblemSolverImpl.solve           action: ScheduleJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchResourcesAction isPost: false
INFO ProblemSolverImpl.solve           action: MatchSchedulerAction isPost: false
INFO ProblemSolverImpl.solve           action: ScheduleJobAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 4 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AcceptMsgAction
INFO CIRAgentJADE.execute       MatchResourcesAction
INFO CIRAgentJADE.execute       MatchSchedulerAction
INFO CIRAgentJADE.execute       ScheduleJobAction
INFO AcceptMsgAction.action received null; blocking
INFO AcceptMsgAction.action received msg topic: matching
INFO MatchResourcesAction.action found resource provider: e1605183-8e76-11e7-bb66-00e0817406be
INFO MatchSchedulerAction.action found scheduling agent: simpleschedule
INFO ScheduleJobAction.action submitting job for topic: simpleschedule
INFO CIRAgentJADE.sendMsg type: ACLMessage.REQUEST topic: simpleschedule

------ SimpleSchedulingAgent.log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> ExecuteJobAction
INFO ProblemSolverImpl.solve InitialStateAction -> ExecuteJobAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> ExecuteJobAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: SimpleScheduleAction isPost: true
INFO ProblemSolverImpl.solve           action: ExecuteJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> SimpleScheduleAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: true
INFO ProblemSolverImpl.solve           action: SimpleScheduleAction isPost: false
INFO ProblemSolverImpl.solve           action: ExecuteJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: SimpleScheduleAction isPost: false
INFO ProblemSolverImpl.solve           action: ExecuteJobAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 3 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AcceptMsgAction
INFO CIRAgentJADE.execute       SimpleScheduleAction
INFO CIRAgentJADE.execute       ExecuteJobAction
INFO AcceptMsgAction.action received null; blocking
INFO AcceptMsgAction.action received msg topic: simpleschedule
INFO SimpleScheduleAction.action scheduled resource: e1605183-8e76-11e7-bb66-00e0817406be
INFO ExecuteJobAction.action submitting job for topic: execute
INFO CIRAgentJADE.sendMsg type: ACLMessage.REQUEST topic: execute

------ JobExecution.log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> SubmitMsgAction
INFO ProblemSolverImpl.solve InitialStateAction -> SubmitMsgAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> SubmitMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: true
INFO ProblemSolverImpl.solve           action: SubmitMsgAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: SubmitMsgAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 2 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AcceptMsgAction
INFO CIRAgentJADE.execute       SubmitMsgAction
INFO AcceptMsgAction.action received null; blocking
INFO AcceptMsgAction.action received msg topic: execute
INFO SubmitMsgAction.action submitting job for topic: e1605183-8e76-11e7-bb66-00e0817406be
INFO CIRAgentJADE.sendMsg type: ACLMessage.REQUEST topic: e1605183-8e76-11e7-bb66-00e0817406be

------ ResourceProvider.log ------
INFO CIRAgentJADE.nextState null -> InitialStateAction
INFO CIRAgentJADE.execute InitialStateAction -> ExecuteJobAction
INFO ProblemSolverImpl.solve InitialStateAction -> ExecuteJobAction requires true
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> ExecuteJobAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: false
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: true
INFO ProblemSolverImpl.solve           action: ExecuteJobAction isPost: false
INFO ProblemSolverImpl.solve       nextGoal: InitialStateAction -> AcceptMsgAction
INFO ProblemSolverImpl.solve           action: InitialStateAction isPost: true
INFO ProblemSolverImpl.solve           action: AcceptMsgAction isPost: false
INFO ProblemSolverImpl.solve           action: ExecuteJobAction isPost: false
INFO ProblemSolverImpl.solve       loop count: 2 found: true exhausted: false
INFO CIRAgentJADE.execute adding required actions
INFO CIRAgentJADE.execute       AcceptMsgAction
INFO CIRAgentJADE.execute       ExecuteJobAction
INFO AcceptMsgAction.action received null; blocking
INFO AcceptMsgAction.action received msg topic: e1605183-8e76-11e7-bb66-00e0817406be
INFO ExecuteJobAction.action sent job request to cds-station2.eng.uwo.ca
```

Curriculum Vitae

Name:	Adrian T. Bienkowski
Post-secondary	The University of Western Ontario
Education and	London, Ontario, Canada
Degrees:	1993-1998 B.Sc – Electrical Engineering.
	The University of Western Ontario
	London, Ontario, Canada
	1993-1997 B.Sc. – Computer Science
Related Work	Research Assistant
Experience	The University of Western Ontario
	2012-2017

Publications:

[1] Adrian Bienkowski, Hamada H. Ghenniwa (2017)
Extending Network with Resource Brokering for Grid Environments, (in-progress)

[2] Afshan Samani, Adrian Bienkowski, Raafat Aburukba, Hamada H. Ghenniwa (2014)
Privacy protection management in Open CDS Environments, (in-progress)

[3] Afshan Samani, Raafat Aburukba, Adrian Bienkowski and Hamada H.Ghenniwa
(2013) Privacy Framework for Open Environments, PASSAT 2013