Meta-brokering solution for establishing Grid Interoperability

Ph.D. Thesis

by

Attila Kertész

Supervisor:
Prof. Dr. Péter Kacsuk
MTA SZTAKI

A Thesis Submitted in Fulfilment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Doctoral School of Computer Science
Faculty of Science and Informatics
University of Szeged

Szeged, 2011

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Péter Kacsuk, my supervisor, for his mentoring, encouragement and for guiding me into the world of Grid Computing by helping me to find my own research topic and supporting me to reach my goals.

my colleagues, especially Zsolt Németh, Zoltán Juhász, Gergely Sipos, Gábor Kecskeméti and Lajos Schrettner, for inspiring me, sharing ideas and for their valuable advices that helped me to publish papers and arrive at this dissertation.

my co-authors, for helping me to gain research skills during my scientific visits to institutes abroad, including Peter Praxmarer, Ivan Rodero and Francesc Guim.

and finally my wife, my family and my friends, for their love and continuous support during my studies.

Ask and it will be given to you; seek and you will find;
knock and the door will be opened to you.
For everyone who asks receives; he who seeks finds;
and to him who knocks, the door will be opened.
Matthew 7:7-8

Chapter 1

Introduction

Grid Computing [29] has become a separate research field in the '90s and since then it has been targeted by many projects all around the world. Several years ago users and companies having computation and data intensive applications looked sceptical at the forerunners of Grid solutions that promised less execution time and easy-to-use application development environments by creating a new virtually unified high performance system of interconnected computers from all around the world. Research groups were forming around specific parts of Grid systems and different research areas emerged, because former techniques of distributed computing were not applicable in Grid systems. Many user groups from various research fields (biology, chemistry, physics, etc.) put their trust in Grids and today usage statistics and research results show that they were undoubtedly right. Grid Computing has been in the spotlight, several international projects have aimed to establish sustainable Grids (eg. CoreGRID [95], EGEE [92], NextGRID [121], GEANT [97], KnowARC [104], EUAsiaGrid [94] and OSG [127]).

Core Grid services are provided and implemented by a so-called Grid middleware [33]. The first widespread middleware was the Globus Toolkit [30], which became a de facto standard for Grid Computing around 2002. Since then several middleware solutions have appeared, and the production Grids using these solutions formed separate islands that represent borders for both researchers and user communities. A decade of Grid development has established many national and international production Grids based on different middleware solutions (eg. HunGrid [110], NGS [120], EGEE [92], UNICORE [143], NorduGrid [122] and OSG [127]). As a result of the numerous Grid projects and available production Grids, user support centers [142, 98, 146, 103]

have been set up in order to ease application porting to Grid environments. In some cases these applications are so large and complex that their executions require more computing resources than a particular Grid can provide. Therefore similarly to the World-Wide Web, the interconnection of these separate islands can result in a World-Wide Grid in the future. Such an aggregated system could cope with the growing number of users and computation-intensive applications.

Resource management in Grid systems is the research field most affected by user demands. Though well-designed, evaluated and widely used resource managers (also called as brokers) have been developed, new capabilities are required, such as interoperability and agreement support. The available resource managers have already been surveyed by other research groups [52], but these publications do not detail capabilities related to interoperability and do not separate operational roles (eg. scheduling, brokering, management). This dissertation aims at providing a high-level brokering solution to establish Grid Interoperability [70], which means the bridging of different Grid infrastructures in order to allow users on one Grid to run computing jobs and exchange data with users on other Grids. The current solutions of Grid resource management will not be able to fulfil the high demands of future generation Grid systems, though several Grid resource brokers [2] have been developed supporting different Grid systems. The main problem is that most of them cannot cross the borders of separate Grid islands caused by different Grid middleware solutions, therefore they can mature as slowly as middleware solutions evolve. These newly arisen problems need to be treated by novel research approaches in order to aggregate the separated Grid islands and manage them together, because currently used Grid middleware solutions do not support real interoperation other then restricted bilateral ones.

Solving these problems is crucial for the next generation of Grids, which should spread from the academic to the business world. The advance of Grids seems to follow the way foreseen by the Next Generation Grids Expert Group, which has been established by the European Commission. In their third report [61] they have pointed out that Grid and web services are converging and envisaged hybrid services called as SOKUs (Service Oriented Knowledge Utility), which enable more flexibility, adaptability and advanced interfaces, therefore interoperability is evident and congenital in these systems.

Following these expert guidelines and the latest requirements of Grid user groups, I propose in this dissertation such a high-level Grid brokering solution that enables Grid Interoperability by providing the highest number of brokering capabilities in a

way that it does not require any changes to the underlying Grid middleware services.

1.1 Summary of research results

During the research presented in this dissertation my first goal was to elaborate a classification of Grid resource brokers. At that time, the less than ten-year-old Grid Computing had several resource management solutions named by different expressions operating on different middleware addressing various user needs. During the preparation of the first thesis I examined the widespread Grid resource brokers used by different user communities, identified their key functionalities and properties, gathered them into a taxonomy, and classified them in a survey using the elements of the taxonomy. I analysed the connections and inner structures of the available Grid resource manager components, identified different operational roles and resolved their contradictory naming acronyms and expressions by creating an anatomy of Grid resource managers. I formalized the identified brokering roles, and inserted them into the Abstract State Machine (ASM) model of Grid systems [60]. I identified and defined interoperability levels for Grid brokering solutions and expressed them in the presented model that enables the classification of related brokering approaches. I stated the following thesis based on these results:

> **Thesis I. I designed a category framework of broker capabilities that I used to create a general taxonomy of Grid brokers. I designed an anatomy of Grid resource managers that I used to formalize Grid brokering levels based on the ASM model of Grids [60].**

Grid Interoperability [70] is a fundamental challenge of Grid Computing nowadays. The presented broker taxonomy also points out the heterogeneity in most brokering components and methods. The resource management anatomy revealed their similarities and possible interactions that paved the way for introducing a meta-level in Grid brokering to interoperate different Grid systems. Some of the surveyed brokers are capable of low-level interoperation by accessing resources of different Grids. I showed how these approaches address multi-grid brokering by broker-extension and multi-brokering from Grid portals. For a higher level of interoperability, a general broker description language is needed in order to enable the unified management of Grid brokers. The second thesis contains the elaboration of such language based on a meta-data model, using the categories of the broker taxonomy.

Thesis II. I designed a new, XML-based description language called Broker Property Description Language (BPDL) that is able to describe any Grid resource broker that can be categorized in the taxonomy. A high-level brokering service can use this language for the unified management of these brokers.

I named the novel approach that performs high-level brokering at the meta-level of Grid resource management as meta-brokering. The next, third thesis includes the description of the required components of a general meta-brokering architecture (besides the broker description language) and a realization of the abstract architecture in a meta-brokering service that does not require any modifications to the utilized brokers and Grids.

Thesis III. I determined the general requirements of Grid meta-brokering, and developed a general architecture based on these requirements that introduces a higher abstraction layer for enabling Grid Interoperability by the unified management of Grid brokers. Based on this general architecture, I designed the necessary components to build the Grid Meta-Broker Service (GMBS).

The components of the realized meta-brokering service perform user interactions, monitoring of resource and Grid load, tracking broker performance and automatic broker selection. After publishing this meta-brokering approach, other research groups have also realized the need for interoperable brokering and started to develop their own solutions. I designed a classification of these solutions based on the interoperability levels introduced in Thesis I. The final part of the research was to evaluate the proposed meta-broker. The GridSim Toolkit [12] is a widely accepted and used Grid simulator that can be easily tailored to analyse Grid brokering methods. The fourth thesis presents a meta-brokering simulation architecture that extends GridSim, and the performance evaluation of the implemented meta-broker in this environment by using real world resource usage traces form the publicly available Parallel and Grid Workloads Archive [129, 107].

Thesis IV. I developed a new simulation environment based on the GridSim [12] simulator that is able to evaluate meta-brokering. I performed the evaluation of GMBS in this environment with a performance analysis using both real parallel and Grid workload traces.

I proved the effectiveness of the interoperable meta-brokering service with the evaluation.

The evaluation results showed that the interoperable meta-brokering solution of GMBS was able to achieve an order of magnitude better performance in Grid application execution compared to the general, non-interoperable Grid utilization simulated by random broker selection.

Table 1.1 shows the connections between the theses and the publications:

Table 1.1: Theses and publications

	[P4]	[P18]	[P16]	[P1]	[P5]	[P17]	[P2]	[P11]	[P6]
Thesis I.	•	•	•	•	•	•	•	•	•
Thesis II.						•			•
Thesis III.		•	•			•			•
Thesis IV.						•			

	[P7]	[P3]	[P8]	[P10]	[P14]	[P19]	[P12]	[P9]	[P13]	[P15]
Thesis I.	•					•				
Thesis II.	•		•	•	•	•		•		
Thesis III.	•	•	•	•	•	•	•	•	•	•
Thesis IV.						•			•	

1.2 Agenda

This section provides an outline of the dissertation to show where and how it validates the claims made previously.

Chapter 1 contains the introductory part of the research performed including the theses of the dissertation followed by an overview of the structure of the document itself.

Chapter 2 and 3 describes the problem area starting with a motivation example followed by a literature review and a deep investigation of Grid resource managers. Based on the presented findings a formal model has also been developed that serves as a basis for the comparison of related research approaches. The contributions of Thesis I. are discussed in these chapters.

Chapter 4 provides the novel solution in the area of Grid resource management for the problem of Grid Interoperability. It also enumerates avenues of future work for further development of the concept and its applications. The contributions of Thesis II. and III. are discussed in this chapter.

Chapter 5 is where the experiments, the evaluation of the proposed novel meta-brokering solution are fully described. This part includes the details of how the empirical side of the research has been conducted. The contributions of Thesis IV. are discussed in this chapter.

Chapter 6 contains a restatement of the claims and results of the dissertation.

Appendix A and B contain summaries of the dissertation in English and Hungarian.

Appendix C contains additional technical information related to the broker description language and the implementation of GMBS introduced in Chapter 4.

Chapter 2

Problem statement

The EGEE project [92] has been initiated in Europe to target two well-defined application areas: high energy physics and biomedicine. In high energy physics, *very large amounts* of data are produced and analysed, therefore it has one of the key user groups running applications on the EGEE infrastructure. Since then a wide range of research areas has appeared including Earth Sciences, Astroparticle Physics, Computational Chemistry, Drug Discovery, Hydrology and Cosmology. EGEE formed a strategic alliance with the LHC Computing Grid (LCG) project [148] from the beginning, in order to satisfy the computing needs of the Large Hadron Collider (LHC) particle accelerator in CERN, which is one of the largest scientific instruments in the world planned to provide 15 petabytes of data per year. Besides, four different LHC experiments also use EGEE resources: ALICE (A Large Ion Collider Experiment), ATLAS (A Toroidal LHC Apparatus), CMS (the Compact Muon Solenoid Experiment) and LHCb (The Large Hadron Collider Beauty Experiment) [92].

The biomedical area has also joined the EGEE project in the beginning. It has various applications for drug discovery, medical tomography, protein analysis, molecular docking analysis and mass screening of molecular interactions [92]. As a result of the numerous Grid projects and available production Grids mentioned in the introduction, user support centers have been set up in order to *ease* application porting to Grid environments. The TeraGrid Advanced User Support [142] (AUS) project serves American research communities, while in Europe the Global Grid User Support [98] (GGUS) portal and the Westminster Grid Application Support Service (W-GRASS) [146] provide application porting services. In Hungary, the Grid Application Support Centre [103] (GASuC) provides similar facilities.

As a result of these user support teams, *numerous* applications have been gridified that require *enormous* computational power. For example, one of the 1195 applications published in the myExperiment Project [119] that can be executed in EGEE [92], the Success Abandonment Classification complex Taverna workflow by Andrea Wiggins [140] contains 65 tasks shown in Figure 2.1. It retrieves data from FLOSSmole and from the Notre Dame SourceForge repository to compute project statistics based on releases, downloads and project lifespan. These statistics are then used to classify projects according to different comparison criteria.

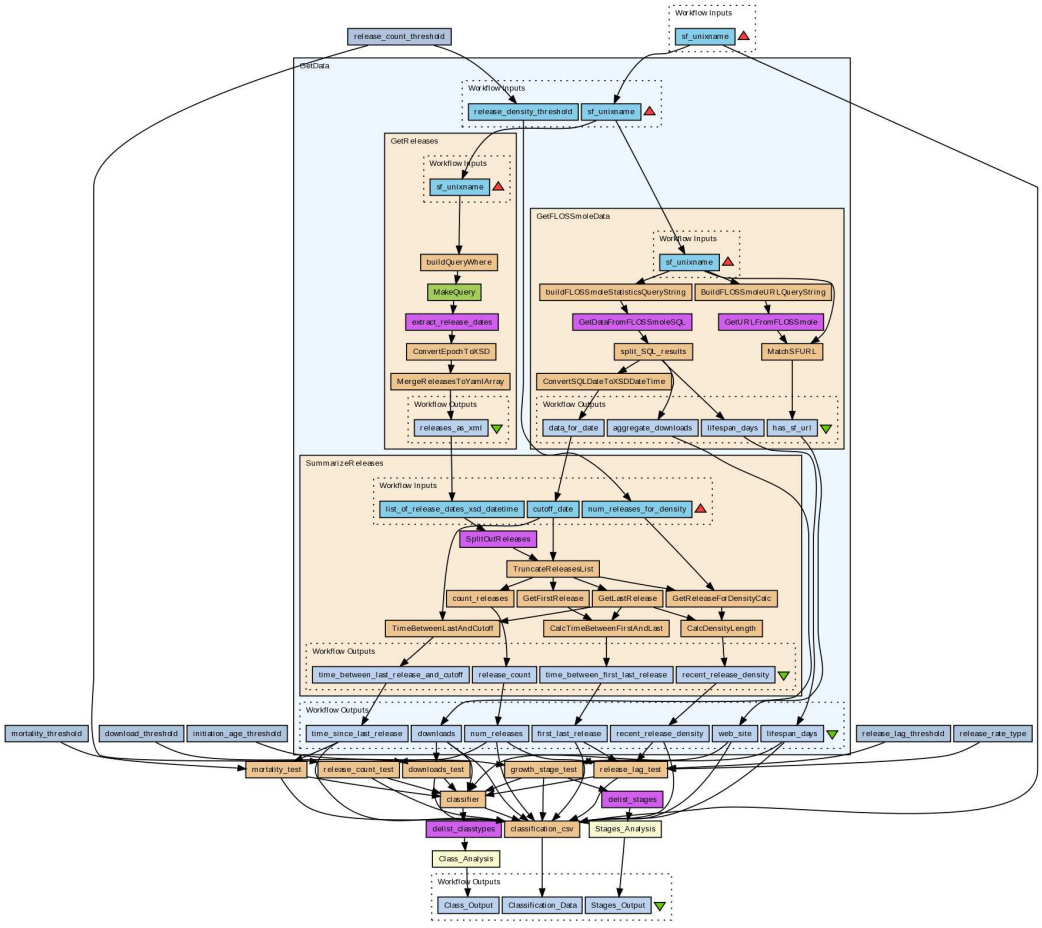

Figure 2.1: A complex Taverna workflow

The UK ProSim project has a proteine molecule simulation workflow [134] for in silico modelling of intermolecular recognition, which is critical for biological processes in human cells including the function of antibodies in immune responses to invading pathogens and xenotransplants. The application uses in silico modelling for determining how proteins interact with ligands and how to manipulate them to improve

or change their specificity. It integrates readily available software programs in an optimised workflow to reproduce receptor-ligand complexes with a good degree of accuracy. One single Lysozm molecule simulation of the application alone costs 170 CPU hours. The UK National Grid Service [120] (NGS) is used to decrease execution times of these experiments.

The Savannah Experiment [135] application by A. Lynch examines the sensitivity of the Australian monsoon to savannah fire. Each 21 year simulation takes about 6-12 weeks to execute on a single processor. The gridified workflow separates each simulation into 252 monthly steps. In the project the first experiment was carried out on 7 hosts of the PRAGMA Grid [131]. It contained 90 simulations running for 143 days. They plan to perform a second experiment involving TeraGrid [142] and OSG [127].

The Phaser experiment [133] examined the Phaser molecular replacement (MR) application by A. M. Buckle that uses a brute force approach to identify candiate models for MR which involves exhaustive MR calculations using representative structures from a database. They used OSG [127], PRAGMA [131] and GIN-VO [125] resources in order to demonstrate the need for interoperability. The experiment took for two months executing around 71,000 jobs using 511,000 CPU hours.

Biomedical research is also a highly studied area in Hungary. For the request of a researcher of the Biological Research Center (BRC) of the Hungarian Academy of Sciences I have ported a biochemical application [47] to EGEE that also serves as a motivating example for the research aims of this dissertation. This application generates 50 000 conformers of flexible molecules by unconstrained molecular dynamics at high temperature to overcome conformational bias, then finishes each conformer by simulated annealing and energy minimization to obtain reliable structures. The execution time of this application for one particular parameter-setup on a single machine takes around one week, while I have managed to reduce it to one day by running the gridified application in one of the Virtual Organizations (VO) of EGEE [47]. When this application is used in production, at least one quantitative structure-activity relationships (QSAR) study is executed, which needs 15-20 molecules as a minimum to simulate. This implies that for a complex statistical study hundreds of such an application need to be executed, which cannot be performed even within some months time on a single Virtual Organization.

As we have seen, in several cases these applications are so large and complex that the execution requires *more computing resources* than a particular Grid can provide.

In order to produce results for similar complex applications within a time frame acceptable for researchers, the different Virtual Organizations and separate Grids need to be *managed together*, need to be aggregated. Such a virtually unified system could cope with the growing number of users and computation-intensive applications.

2.1 The role of Grid brokering

Executing a user application in a Grid environment requires several prerequisites. Users need to learn the interfaces of the Grid services and need to describe their application prior to submissions. Production Grid systems may consist of hundreds of thousands of resources (eg. 240,000 processor cores in EGI [93]), therefore it is not an easy task to find out the actual state of the computing and storage resources and *select* one for a user program. There was no question about automating resource discovery and selection. Special resource managers, also called resource brokers are meant to solve this problem [2]. As resource management is a key part of current Grid middleware solutions, and most middleware developer groups and projects have developed their own tools for resource brokering.

Job scheduling on a multiprocessor system has been studied for more than 30 years and is known to be *NP-complete* [83]. Scheduling in Grid systems, which is one of the tasks of Grid resource managers, become more complicated with multi-organizational shared resources, therefore Grid scheduling is also NP-hard [35, 74]. In order to achieve better scheduling the general approach is to use some form of heuristics, eg. job run time estimates. On the other hand, the inaccuracy of these estimates is a perennial problem mentioned in the job scheduling literature, and even if the users are required to provide these values, there is not a substantial improvement in the overall average accuracy [56]. In [69], Ramirez-Alcaraz et. al. have analyzed different Grid allocation strategies depending on the type and amount of information they require, and they found that the information about user run time estimate and local schedules does not help to significantly improve the outcome of the allocation strategies. They concluded that quite *simple* schedulers with minimal information requirements can provide *good performance*. Practice seems to adapt to these findings, because too complex, sophisticated scheduling algorithms are rarely used in Grid brokers, as we will see in the next section.

To enhance the manageability of Grid resources and users, Virtual Organizations (VO) were founded. This kind of grouping has also started an isolation process in

Grid middleware development forming separate islands in the ocean of Grids. *Interoperability* among these islands plays an important role in current Grid research. This chapter gives a classification of the present Grid resource brokers by their relevant properties and functionalities. Identifying the key features and mapping them to user needs can open a new way for enhancing interoperability among different Grids. Although the same services are available in different middleware, they have been implemented in different ways. This taxonomy gives an insight how these brokers are built up and can be accessed, and helps researchers to have a better understanding of the current trends of resource brokering.

2.2 Taxonomy of resource brokers

Regarding taxonomies in Grid Computing, two main papers have been published about resource management systems [52] and workflow management systems [88]. As a resource broker is usually part of the resource management system (RMS) of a Grid, the first one is closer to my particular research area. That taxonomy introduces an abstract model of resource management in different Grid Systems, then describes and compares the existing architectures. While each RMS operates on one middleware, resource brokers can be middleware-independent entities, therefore some of them are able to access resources of different middleware. A more concrete distinction and clarification of different resource manager components will be given in the next chapter in Section 3.1. This taxonomy is needed to *clarify* the role and usage of current resource brokers, and to gather and present also those ones that were out of the scope of the RMS taxonomy. I further examine the interfaces and the implementation details of these brokers to reveal their main capabilities and properties.

The aim of this *taxonomy* is to gather the recent Grid brokers used by different Grid user communities, highlighting their main properties and examining the differences and similarities regarding their architecture and operation. I classified the revealed properties to 7 major categories and split into three groups. The following subsections comment the categories of these groups.

The *first* group is middleware oriented (Middleware Support), the *second* explains mainly the user application-related categories (Interface, Job Model, QoS and Data Movement), finally the *third* deals with scheduling features (Information System Support and Scheduling Model). The simplest, typical user application in Grids is called a job. A complex user application called a workflow can be built up by more in-

terconnected jobs, where the interconnection represents data or control dependences among the jobs. In this dissertation I assume that user applications are submitted to the system (to the broker) in the form of jobs, or in case of workflows a higher level service (eg. a workflow enactor or manager in [88]) submit the jobs of the workflow to the system in a proper execution order.

2.2.1 Grid Middleware

The first main category – shown in Figure 2.2 – shows the underlying infrastructures of the overviewed brokers. They usually rely on one of these middleware solutions [99, 100, 25, 77] and use their functions to discover resources and submit user jobs. We can distinguish between service-based and non-service-based ones. Generally this property determines the architecture of the broker. It can be stated that the most widespread middleware is the Globus Toolkit, since LCG-2 is built upon Globus services and the NorduGrid ARC also uses and extends some of them.

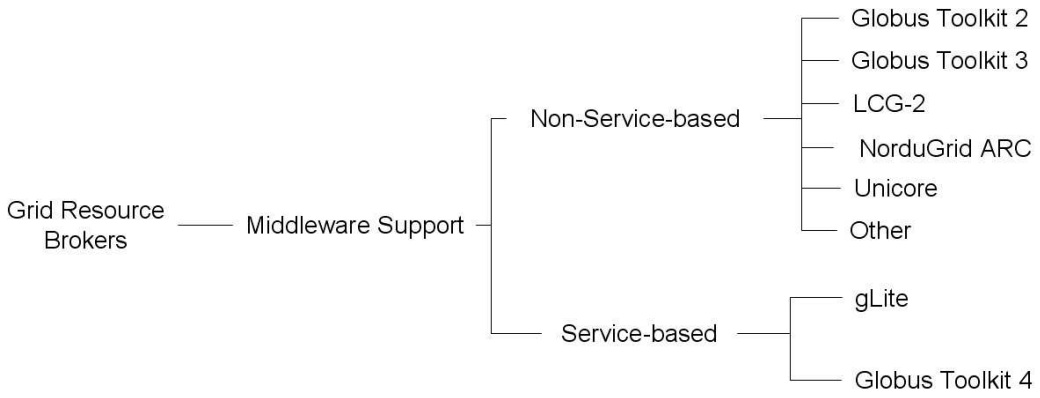

Figure 2.2: Categories of the Taxonomy: Grid Middleware group

2.2.2 Job handling

This group contains mainly user and job related properties and can be seen in Figure 2.3. The first thing the user faces is the interface of the broker. Early solutions provided only command-line access, while APIs are important for higher level utilization and management by other applications. Some brokers even have Graphical User Interfaces (GUI) to ease user interaction. Service-based brokers offer service access (eg. Web-Service interface of VIOLA MetaScheduling Service (MS) [87] and eNanos [71]),

which is an advanced method and needed by the latest developments. This function can enhance interoperability and provide platform-independent access.

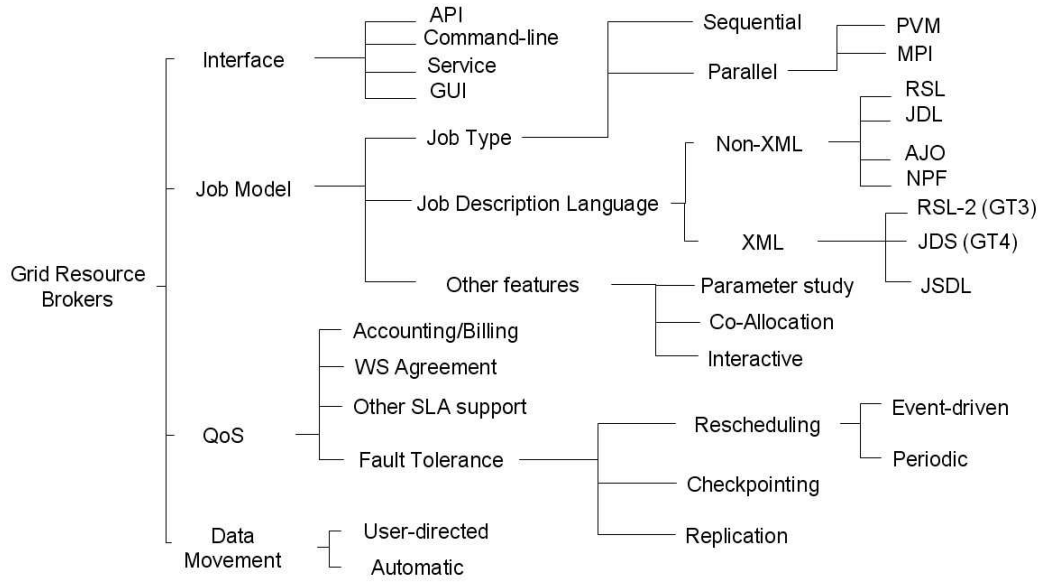

Figure 2.3: Categories of the Taxonomy: Job handling group

The job model of the broker is also important for users and applications. These properties tell how to describe a user job and which types a broker can handle. There are several non-XML language descriptions, but the latest developments follow the XML syntax. It would be reasonable for the brokers to accept and use XML job descriptions, even if they access middleware solutions supporting different languages [99, 92, 25, 77]. In this case they would need to translate the request, but this approach leads to better interoperability. The rest of the properties in this subcategory shows what type of jobs can be submitted to a specific broker: only sequential or parallel; in the second case co-allocation and advance reservation are handled or not. Brokers can support other special job-handling functionalities such as parameter study and interactive jobs.

Fulfilling user requirements is a critical task of the broker. Related properties are gathered into the Quality of Service group. Accounting is used for the administration of the users and tracking their Grid utilization, and billing serves Grid economy. Agreements are used to guarantee some level of service during brokering. User requirements can contain special requests, which are crucial for the job or application. On the other hand resource providers would protect sites from being flooded by user jobs. In order to find a balance and fulfil requirements these policies appear in the

agreements, which are taken into account in scheduling decisions. Various solutions can be developed to create such service level agreements, but this functionality is still an open issue. Basically two types are used: the WS-Agreement [89, 87] and the USLA [21]. The third part of QoS is fault tolerance. The dynamic nature of Grids lowers the number of successful job submissions. To ensure a higher level of quality, brokers should be fault tolerant. Rescheduling and replication are the basic functionalities, and checkpointing can provide a more reliable brokering, though this is rarely supported, yet. Rescheduling can be event-driven or periodic, and usually choosing a different resource makes sense, retry on the same resource is only a waste of time.

Most of the brokers provide automatic centralized data movement for input and output file staging. User-directed utilization can also be supported, when the user copies files to storage elements and tells the broker to use them.

2.2.3 Scheduling

The third group gathers properties related to resource information, discovery and scheduling. The properties of this group are shown on Figure 2.4. Several resource brokers use the information system of the underlying middleware. In this case the relevant information from the brokers' view is the data store and query. The two main subcategories are the directory-based and service-based implementations. These properties tell us how the brokers access resource data and what kind of information they can use for resource mapping – since this is determined by the information system of the middleware. Some brokers use additional information about the Grid gathered by an information system of their own. Examining historical data (resource availability, job failures, etc.) is one of these approaches. The other type of gathering relies on special agents, which provide information about specific elements of the Grid.

Matchmaking is the major task of Grid brokers. The scheduling properties can qualify brokers and determine the goodness of their decisions. In smaller scope of resources like Virtual Organizations (VO), usually a centralized scheduler component is used to make decisions. In decentralized schedulers the matchmaking process can be split up and queues can be utilized for job requests, or more components can collaborate to utilize a wider range of resources. The first solution is rather used in hierarchical and the second in peer-to-peer architectural models. The decision making can be static or dynamic. When a user fixes a resource for its job, or the

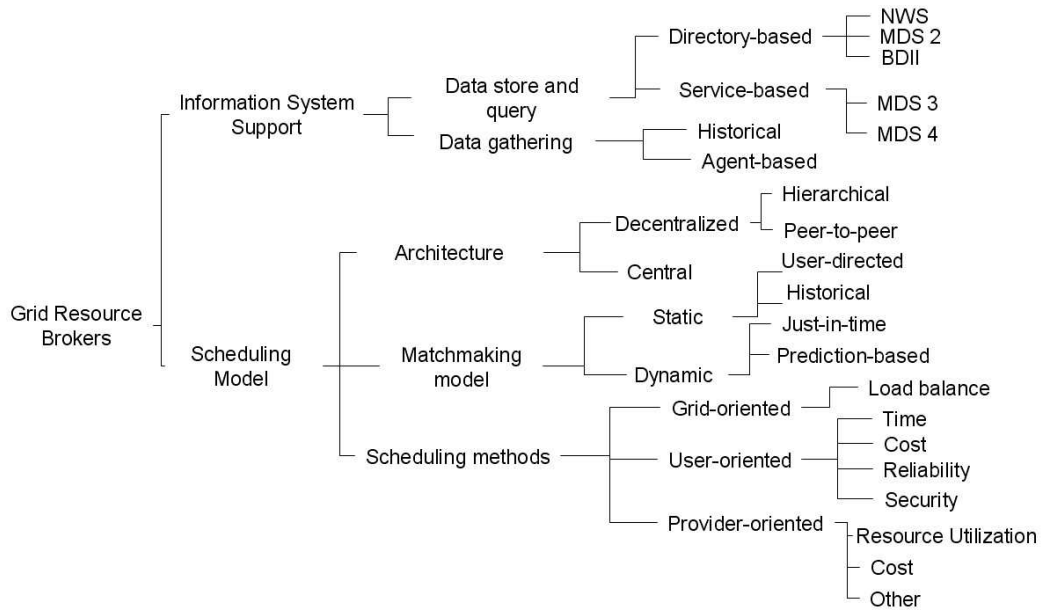

Figure 2.4: Categories of the Taxonomy: Scheduling group

scheduler component of the broker uses only static historical information, we are talking about static matchmaking. In a dynamic decision the broker has an up-to-date information about the resources and makes a just-in-time matching, or uses up some additional prediction-based information. For example, Lőrincz et. al. monitor previous runtime information to determine the behavior of the job and use these additional data in scheduling [58]. The schedulers can take into account specific policies that affect decision making. These methods usually favor the users, but the provider expectations or the balanced state of the Grid can also be observed. User policies can tell the broker to submit the job to a resource that completes the request in the shortest time or for the less cost possible. An example for the time-based user policy is the VIOLA MS [87], which uses a first fit reservation policy that tries to place the job at its requested time, otherwise it schedules the job for the earliest possible time after the one requested. Reliable resource selection can also be an interesting point of view, where less error can occur, or a secure one that ensures the safety of the job. Providers may expect from the broker to utilize more or less a specific resource, or gain as much as they could from the resource utilization. An alternative method is to serve the user requests as to keep the balance of the load on the Grid.

2.3 Survey of resource brokers

The properties of the taxonomy were gathered from *16 Grid brokers*. Table 2.1 shows the examined brokers, and gives a short description of their architecture and operation. The columns corresponding to the groups of categories were described in Section 2.2. This survey displays the main properties of the brokers. It indicates how the categories of the taxonomy are implemented and used in different solutions.

From the survey and the taxonomy we can clearly identify which properties are used rarely and which ones are highly supported. Regarding the whole taxonomy I can state that the Globus Toolkit is used by most of the brokers, therefore the RSL language is still the most widespread. The command-line interface is usual, and most of the brokers use a central scheduling architecture with just-in-time matchmaking optimized for minimal completion time. Rescheduling is widely used for fault tolerance.

On the contrary, the JSDL [113], which is a uniform standardized language, is rarely supported, yet. APIs, co-allocation, advance reservation and interactive job support should be provided by more brokers. A decentralized architecture could be a better solution in several cases, and local information systems should be built to gather more dynamic data and perform prediction-based matchmaking. As Grids are heading towards the markets, provider-oriented policies should be more supported, and economy-based scheduling need to be considered. This solution requires QoS, so agreements must be supported by future brokers. To enhance reliability, checkpointing and job migration should be targeted by future developments. Finally the most important thing to do is to provide all these broker properties to the users, making available more brokering services, more middleware functionalities and more resources in a transparent way. Interoperability is the key to achieve this vision.

The presented taxonomy helps in *identifying and categorizing* the most important properties of Grid resource brokers in various Grid environments. I revealed the interfaces and relevant functionalities of the currently used brokers, which can enhance better resource utilization and future development. With the presented survey users and scientists can have a better understanding of the operation and utilization of the current brokers. Developers should target issues that are missing or rarely used in these solutions, but there is a definite need for them – to achieve this, the properties of the taxonomy give the guidelines.

Table 2.1: Survey of Grid brokers

Grid Broker	Middleware Support	Job handling	Scheduling
AliEn RB [73]	Alice	File transfer optimization, fault tolerance by multithreading	Push and pull task assignment
Apples [15]	GT 2	Parameter study support, event-driven rescheduling	Centralized adaptive scheduling with heuristics, self-scheduled workqueues
eNanos [71]	GT 2, 3	Web-Service interface, API, rescheduling	User-oriented policies, local information system
EZ-GRID Broker [79]	GT 2, 3	GUI for job handling, transparent file transfer	Own information service with dynamic and historical data, Policy Engine Framework for provider policies
GRIDBUS Grid Service Broker [86]	GT, UNICORE, Alchemi	Failure management and application recovery, parameter study, API support (XPML description file)	Economy-based and data-aware scheduling
GridWay [38]	GT, gLite, NorduGrid, OSG	Job migration support (checkpointing, resubmission), API support	Decentralized (or centralized) scheduler, adaptive scheduling
GRIP Broker [9]	GT 2, 3, UNICORE	Ontology Engine for translating different job description	Ontology Engine for translating different information service data
GRMS [53]	GT 2, 3	Service-based, job migration support	Own job description language (GJD), job registry, multicriteria framework
GRUBER [21]	GT 3, 4	SLA-based resource sharing in multi-VO environment, disk quota considerations	Internal site monitoring feature, various user-oriented policies

GTbroker [P1]	GT 2, 3, LCG-2	Periodic and event-driven rescheduling, automated file staging	User-oriented policies, additional dynamic information
JSS RB [23]	GT 4, NorduGrid ARC	WS-Agreement for advance reservations, resource filtering by user requirements, file staging and replication support	Scheduling algorithms with benchmark-based execution time and transfer time estimation
KOALA [59]	GT 2, 3	Periodic and event-driven rescheduling, parallel co-allocated job handling, automated file staging	Processor and data co-allocation, own information service, hierarchical scheduling with queues, incremental claiming policy
LCG-2/ gLite Broker [92]	LCG-2/ gLite	Periodic and event-driven rescheduling, interactive job support	Eager or lazy policies, push and pull models for task assignments, provider-oriented policy support
NIMROD/G [11]	GT 2, Legion	Application level accounting, parameter study support, periodic rescheduling (Nimrod/G plan file)	Deadline and budget-based constrained scheduling, hierarchical and decentralized agent-based scheduler
OGSI Broker [51]	GT 3	User defined ranking in resource selection	User-oriented and provider-oriented resource owner policies, internal agent-based information system
VIOLA MS [87]	UNICORE	WS-Agreement for advance reservations, co-allocation, WS interface	First fit reservation

2.4 The problem of Grid Interoperability

Before discussing the proposed solution of this dissertation, I need to clarify the targeted problem area. My interpretation of *Grid Interoperability* will be given at the end of this section, but before I do that I review the existing definitions that have been stated so far starting from the very general ones. The Oxford Advanced Learner's Dictionary [128] defines the word "interoperable" as:

> *"interoperable – (technical) (of computer systems or programs) able to exchange information"*

Wikipedia [147] defines "interoperability" as:

> *"Interoperability is a property of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, without any restricted access or implementation."*

The definition of the IEEE Glossary [39] for "interoperability" is as follows:

> *"The ability of two or more systems or components to exchange information and to use the information that has been exchanged."*

Finally, in the Information Technology Vocabulary [111] we can found that interoperability is:

> *"The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units."*

Summarizing these definitions I can conclude that an interoperable system should have a commonly accepted interface in order to exchange information and should be able to work together with other systems. Now focusing on the particular research field of Grid Computing, let me gather the definitions from related research papers.

Riedel et. al. have published the view of the Grid Interoperation Now (GIN) Community Group [125] of OGF [124] on Grid interoperation in [70]. They noticed that more world-wide domain-specific Grid infrastructures have emerged orthogonal to national Grid initiatives, and the technology used in these Grids is typically not interoperable with each other. They differentiate between Grid interoperation and interoperability. Their definitions are as follows:

> *"Interoperation is what needs to be done to get production Grids to work together as a fast short-term achievement using as much existing technologies as available today."*
>
> *"Interoperability is the native ability of Grids and Grid technologies to interact directly via common open standards in the future, which is a rather long-term achievement."*

To facilitate interoperation, different translators and adapters should be provided to unify common-purpose components of various Grids. According to [70], GIN addresses interoperation in five specific areas: (i) authorization and integrity management, (ii) data management and movement, (iii) job description and submission, (iv) information services and (v) cross-grid operations on multiple Grid infrastructures. For example the GIN-INFO area of GIN plans to unify monitoring information into a common database (GIN-BDII), or regarding data management the GIN-DATA area targets interoperation of data transfers among different Grids. The GIN-JOBS area deals with job submission interfaces and proposed JSDL [113] for a unified job description and OGSA-BES [96] for a generic submission interface. Besides these definitions, the GIN guidelines do not include any attempt to provide a common allocation protocol or brokering solutions of resources between production Grid projects and infrastructures. They say this is beyond the scope of the GIN efforts [125] and resource allocation decisions are left to negotiations between projects or the individual Grid infrastructures. As a summary, they try to develop short-term solutions to support interoperation, but they also keep in mind that these solutions should be revised and standardized to reach interoperability in the long term (once future middleware releases incorporate these standards).

Field in [28] argues that a common interface could solve interoperation among different Grids, but reaching agreement on which interface to use and implementing the selected one by all parties will take time. Since common standards for Grid Interoperability are still being defined and only a few have been widely accepted, they also think that adapters, translators and gateways are needed. Adapters are used to bridge incompatible interfaces, and translators are used to convert information to a format other systems can understand. To use them some parts of the middleware may have to be modified, but generally they can use their own interfaces. Their usage may also indicate the areas where standardization is needed. A gateway is a service that is independent from the middleware and bridges different Grid infrastructures. It

can be used without the modification of the middleware, but it may become a single point of failure or a scalability bottleneck. He says Grid interoperation is a bilateral activity between two Grid infrastructures, what he exemplifies with the interoperation activity of EGEE and OSG. Finally he states that even with technical interoperability assured, a truly federated Grid will bring additional operational challenges, since Grid infrastructures still evolve.

The KnowARC project [104] has published a survey of 9 Grid middleware solutions [114], in which they identified so-called minimal pre-conditions for Grid Interoperability. Their definition for interoperability is:

> *"The subject of Grid interoperation and interoperability is the bridging of Grid infrastructures, allowing users on one Grid to run computing jobs and exchange data with users on other Grids."*

In this document the authors referred four models (LISI, LCIM, LCI and SOSI) that classify the degree of interoperability among systems or components into interoperability levels, and use the levels of Information Systems Interoperability (LISI) that defines five stages from isolated to enterprise levels. This model rather classifies solutions according to their operational layers: a low-level solution that acts at the Resource layer of Grids (eg. cluster manager) falls into level 2 of LISI, while a high-level solution that acts at the Application layer of Grids (eg. workflow enactor) falls into level 4, which is the highest. I believe that a solution acting at lower layers of Grid systems (or the middleware stack) may have higher degree of interoperability then a solution operating at the highest layer of Grid systems. Since my research focuses on resource management in Grids I will use a different classification scheme in Section 3.4 of the next chapter, in order to compare the degree of interoperability of related solutions. The survey concludes that loose and tight integration can be used to create interoperability among different Grid middleware solutions. In a tight integration they should share common interfaces (this corresponds to the long-term solutions envisioned by GIN), while in a loose integration gateways should be developed (which matches the short-term goals of GIN). I propose solutions for this latest, loose Grid integration in the next chapter.

I mostly agree with the definitions and views stated above, which together represent the *problem of Grid Interoperability*. I also think that Grid Interoperability will be fully achieved in the long-term based on standards, and walking on this path short-term solutions will provide interoperation among different Grids. Nevertheless I do

not want to make a clear distinction between the terms interoperability and interoperation, since solutions performing interoperation are created to serve (the final goal of) interoperability. Out of the five specific areas GIN addresses for interoperation, I neglect authorization and data management in a sense that I let them handled by third-party tools (eg. certificate management of portals). Keeping this in mind, first I summarize the current short-term solutions in the next Section, then in Chapter 3 I further examine the resource management area of Grids to find a more suitable place to establish interoperability, and in Chapter 4 I present my high-level brokering solution for Grid Interoperability that manages to interoperate different Grids by providing the highest variety of brokering capabilities to the user communities.

2.5 Multi-grid brokering approaches for interoperation

To cope with the highly dynamic nature of Grids, end-users typically access Grid resources through resource management systems or Grid portals that serve as both application developer and executor environments. Unfortunately, these tools are typically tightly coupled to one specific Grid environment and do not provide multi-grid support. Even if a tool is connected to multiple Grids, applications that utilize services from these Grids simultaneously are not supported. There have been several attempts to make existing production Grids and Grid services interoperable, but none of them have succeeded to establish a permanent, interoperable Grid environment.

Grid Interoperability can be targeted at *different layers* of Grid systems (see Figure 2.5). At the lowest, Network and Resource layers (where networking tools, operating systems and cluster managers can be found) we can find the highest variety of hardware and software (even under the same Grid middleware solutions), therefore there is no reason to address these layers. The layer of Grid middleware seems to be a better choice. Though there are still many services to be modified, the UniGrids [144] project has developed a solution at this layer, but it operates only between Unicore and Globus Grids [99]. The highest, the Application layer lies between the users and the middleware. At this stage we can develop or redesign high level middleware tools to create reliable and easily manageable connections among different systems: portals and other application development and executor environments belong here. In this dissertation I target the *third and fourth* layers. Before I introduce

a high-level brokering solution to enable Grid Interoperability, I discuss two instant approaches that enhance Grid interoperation: *broker extension* and *multi-brokering from portals*. Though these approaches can establish interoperation among different Grids, there are certain disadvantages that should be eliminated by a sustainable and easily maintainable solution.

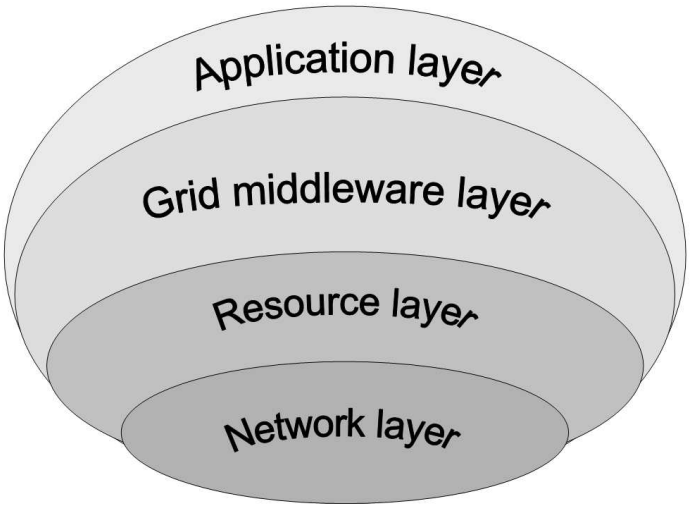

Figure 2.5: The architecture of Grid systems.

2.5.1 Multi-grid extension of GTbroker

The most obvious way to provide interoperability among different Grid systems is to extend the existing and widely used Grid resource management systems (or resource brokers) with multiple Grid middleware support. This approach has several advantages and disadvantages: probably this modification would favour the users most, since they would not need to change their customs, submission methods or job descriptions. But from the other point of view, it requires lots of efforts by the developers to interface new middleware services, so it is definitely a time consuming solution. Furthermore the more systems a broker supports, the more ponderous and unmanageable it becomes. In Section 2.3 we have seen from the taxonomy that such an extension can contribute to Grid Interoperability. As a short-term, but instant solution, I *demonstrate* how this multi-grid brokering can be achieved by describing the extension of *GTbroker* (also considered in the taxonomy) to EGEE Grids [92], which tool has been developed on top of the Globus Toolkit 2 [99], the first widespread and stable Grid middleware.

GTbroker uses an extended RSL (Resource Specification Language [99]) file that should contain the user requirements and job properties. We can use additional attributes (included as comments in the RSL) for specifying the target VO (*voname*) and the location of the input and output files (*jobpath*). Users can influence resource selection by specifying minimum disk space requirement (*mindisk*) and a resubmission deadline (*skip*) that forces job cancel and resubmission, when a job is still pending on a resource after the specified time period. The default scheduling policy computes ranks for available resources according to their queue lengths, and the resource with the highest rank is selected. In addition four predefined scheduling policies can be used (*sched*): resource ranking by CPU speed, disk space or memory size, and the fourth is random resource selection (from a predefined number of resources with the highest ranks). The matchmaking process of the broker uses both static and dynamic information: regarding information systems, the MDS contains static properties of the appropriate Grid resources; to gain dynamic information, GTbroker asks the local schedulers for present availability and the load of each node in the selected cluster. In this way the broker can determine the actual load, right before submitting the job to the selected resource. This additional piece of information makes the broker able to react to dynamic changes, and to avoid choosing an overloaded cluster. With this method, it automatically finds the resources with the highest availability, therefore the submitted jobs can run as early as possible. Fault tolerance is supported by resubmissions. Should a job fail or be pending for too long on a resource (this time interval can be set in the broker), the broker cancels and resubmits it to another high priority one. The actual states of the jobs are tracked by the broker, therefore it is possible to cancel and resubmit jobs.

In order to extend a Grid resource broker to support other types of Grid middleware, first we need to learn how to interact with the new system. Brokers need to gather resource information, move files, perform job submissions, track job states and retrieve output files. Most of these activities need interaction with different middleware services. Security issues such as authentication and authorization could be a challenging problem. In our case all the related Grid systems use GSI certificates, where different proxies can be used for these Grids and they can be treated as other input files. Establishing interoperability in user authorization among different Grid solutions would need high efforts (a specific OGF [124] workgroup is dealing with this topic). Different authorization methods are used in specific Grid middleware (eg. in EGEE and Unicore), furthermore unifying user databases of different Grids/VOs is

not straightforward at all (eg. confidential data handling). Regarding licenses and source code protection, another paper describes a solution called GEMLCA [18] for executing legacy code applications as Grid services. Figure 2.6 shows, how broker-extension can contribute to Grid Interoperability. In the middle we can see the GTbroker, which can be connected to any Globus-based Grid. Generally users submit jobs through its command line interface, which requires the same input data as an ordinary Globus submission (RSL, job and input files). I depicted four Grids, on which the broker was successfully utilized. The Austrian Grid was used for development and preliminary testing. The UK National Grid Service was also used mainly for development purposes. Its Information System contained both MDS2 and BDII attributes, therefore some modifications have been done in the broker. In the following I describe the necessary steps to be taken to extend a broker with interoperable capabilities.

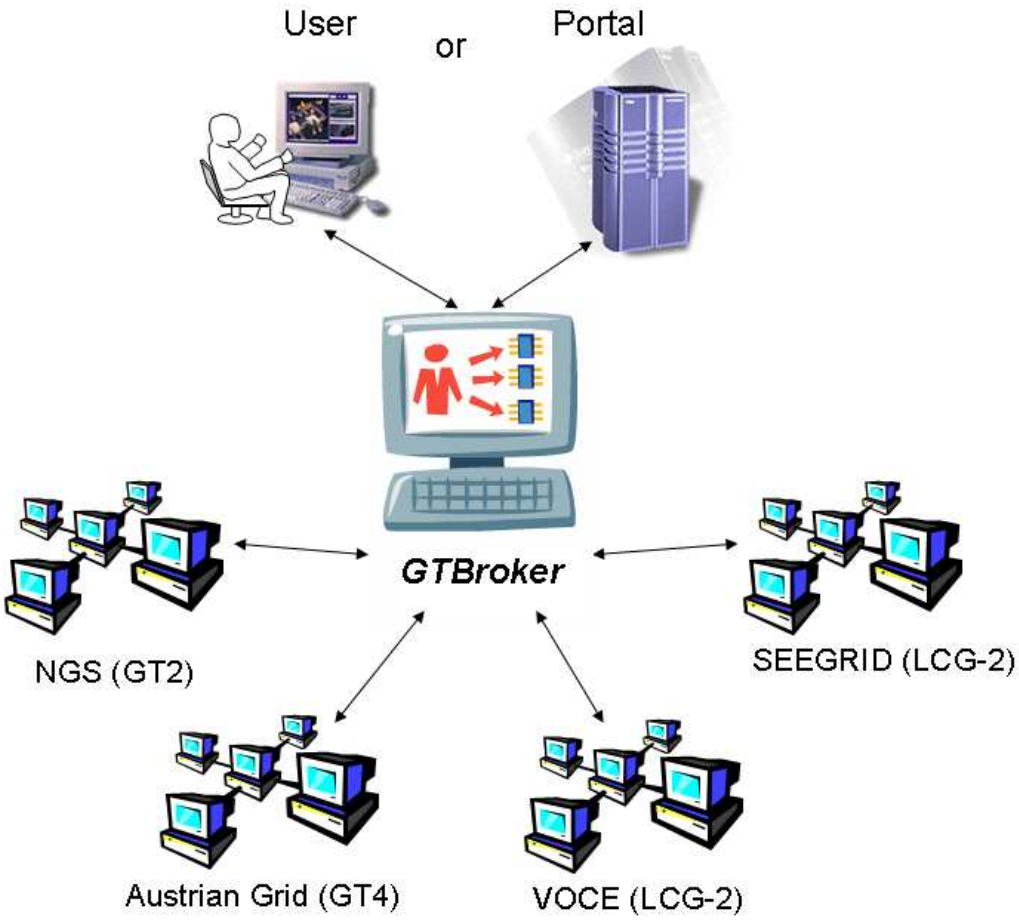

Figure 2.6: GTbroker extension to support Grid Interoperability.

GTbroker has been redesigned to support the LCG-2 (EGEE) middleware, by modifying the information query to be able to gather all data from the information system and handling special attributes in the RSL to enable job submission to EGEE VOs (these VO names have to be specified in general queries and submissions). Since the file movement, job description and job state tracking can also be done through similar Globus services in LCG-2 Grids, I did not modify these parts (nevertheless for an entirely different middleware I should have done it). As a final step of the extension process, I analysed and compared the performance of GTbroker to the LCG-2 Broker (i.e. Workload Management System (WMS) [92]), while they were operated on the same testbeds, under the same load, at the same time. Both brokers mainly rely on the Information System (BDII) of the Grid on which they are utilized. GTbroker orders the resources found in a VO by defining a rank to each of them. The following metrics are used within this calculation: the number of available CPUs in the resource, the maximum number of jobs that can be run on the resource, the number of jobs actually running on the resource, the estimated response time of the resource, and the node count for MPI jobs.

With these metrics the hosts can be ordered in a way that the ones having the best resources for the actual job get higher priority than the others. The LCG-2 WMS also makes decisions by a calculated rank. The default is the estimated response time, and only production state resources are chosen. Furthermore a specific rank can be defined by the user in a JDL [92] description, which is the job specification language in EGEE Grids. In case of data-intensive applications, it tries to find a close host: it takes into account the distance of the physical files on the Storage Elements to the actual Computing Element. I carried out the broker evaluation on EGEE VOs to compare GTbroker and the LCG-2 WMS. I wanted to measure the performance of these brokers and to get some more information on their behaviours, and to demonstrate that GTbroker also performs well on the LCG-2 middleware. The tests contained several phases with different job types. I used the VOCE [145] and SEEGRID [139] production Grids for job submissions. In SEEGRID jobs could be submitted to 18 clusters having 2 to 32 nodes (overall around 130). Regarding VOCE jobs could reach 10 clusters having 2 to 125 nodes (overall around 290). I have used special scripts to utilize the brokers during the whole job life-cycle (multiple submissions, status checking, log retrieving).

For the evaluation I have split the tests into three phases (Table 2.2). In the first phase I submitted 20 jobs to VOCE with short running times and output staging.

Table 2.2: Comparison evaluations between GTbroker and LCG-2 WMS.

Comparison evaluation on production Grids	**Average run time of jobs**	**Number of failed jobs**
VOCE 20 short jobs	GTbroker: 0:01:49 LCG-2 broker: 0:07:39	GTbroker: 0 LCG-2 broker: 16
VOCE 20 long jobs	GTbroker: 0:33:43 LCG-2 broker: 1:43:58	GTbroker: 0 LCG-2 broker: 0
VOCE 20 long MPI jobs	GTbroker: 0:03:26 LCG-2 broker: 0:35:19	GTbroker: 0 LCG-2 broker: 4
SEEGRID 20 long jobs	GTbroker: 0:35:54 LCG-2 broker: 0:37:38	GTbroker: 0 LCG-2 broker: 4
SEEGRID 20 long MPI jobs	GTbroker: 0:39:14 LCG-2 broker: 4:40:31	GTbroker: 3 LCG-2 broker: 3
SEEGRID 60 long jobs	GTbroker: 0:16:31 LCG-2 broker: 0:18:15	GTbroker: 0 LCG-2 broker: 15

With GTbroker I measured 1-2 minutes total run time of all the jobs (makespan), while the jobs with LCG2 broker run for several minutes and most of the jobs failed. I also submitted MPI jobs to VOCE with short running times that run on 5 nodes with job and output staging. GTbroker achieved 3-5 minutes total run time, while with the LCG-2 broker 6 jobs run for 2-10 minutes, the rest took 40-70 minutes or failed to finish. I found that the LCG2 broker picked resources that actually responded after a longer time. In the second phase I also submitted 20 jobs at a time with both brokers to the same VO, but the run time of the jobs took about 10 minutes plus job input and output staging. During one of the experiments most of the VOCE resources were unreliable and only GTbroker made successful submissions with several resubmissions. The total run time of this test took 20-30 minutes, while all the jobs failed with LCG-2 broker and some with GTbroker. Next time I experienced less unreliability but also a heavy load. At this time the average was 30 minutes with GTbroker and 90 minutes with LCG-2 WMS. The SEEGRID VO seemed much more reliable. With the same jobs both brokers achieved 35 minutes average, but some jobs have failed with LCG2 broker. Then I submitted 20 MPI jobs to SEEGRID with 10 minutes subjobs/tasks running on 5 nodes. At this time jobs with both brokers had to wait for clusters with more free nodes so the total run time average was 7.5 hours for GTbroker and 8.5 hours for LCG2 broker, but 9 jobs were failed again with LCG2 broker. I have repeated this measurement, when there was less load on SEEGRID. I measured 40 minutes average with GTbroker and 4.5 hours with LCG-2 broker,

but some jobs were failed for each broker. Finally in the third phase I submitted 60 jobs running for about 10 minutes plus job and output staging to SEEGRID. I started 20 jobs at a time repeated two times after 5 minutes. GTbroker achieved 16 minutes average total run time, while LCG2 broker 18 minutes average with 15 failed or non-responding jobs. I repeated this test several times and measured similar results.

As a summary I can state that sometimes the LCG2 broker makes a random pick and selects slowly responding or even non-responding resources. Resubmissions with the LCG-2 broker were most of the time unsuccessful; therefore it is not as reliable as GTbroker. On the contrary, GTbroker made *reliable* resubmissions and the hidden non-responding or draining resources could have been skipped. For jobs with short running time it produced *better results*, for larger jobs their performances were about the same. Though it has an eager matchmaking, the user can modify the resource selection with the extended RSL attributes (mentioned at the beginning of this subsection). On fairness I remark that both brokers were running concurrently on the related VOs, therefore they were competing for the same resources during the evaluation.

The results demonstrate that existing resource management systems can be extended to use other middleware systems, but in this way developers need to redesign these brokers to support services of the additional middleware. If we take a closer look and examine the existing resource brokers, we find that they also have similar and different properties that may satisfy different user requirements. For example, some of them support co-allocation of parallel jobs, while others provide special fault-tolerant features. When users require most of these features, they still need to use *more* brokers. Next I exemplify, how multi-brokering can be done with the help of portals.

2.5.2 Multi-grid brokering from portals

To exploit the advantages of using various Grids at the same time, we need to utilize more brokers. In this situation we need to learn various job specification languages and broker capabilities. Currently there are several Grid tools available as Grid user interfaces that try to hide the details of low level middleware utilization by providing transparent, uniform access. Grid portals provide a convenient environment for Grid utilization. In this section I show how such a portal can utilize various resource

brokers to access resources of different Grids by providing interoperability among these Grids. There are general purpose and specialized portals for supporting specific applications. For example, the Conflet (CONFigurable portLET) framework [66] can be used to create specific portlets to user applications in portals. In Figure 2.7, we can see how *multi-brokering* can be achieved in portals. In this kind of Grid utilization we do not expect from Grid brokers to support more than one middleware, but to do their best on their initial middleware.

The P-GRADE Portal [46] is a workflow-oriented Grid portal with the main goal to support all stages of Grid workflow development and execution processes. It enables the graphical design of workflows created from various types of executable components (sequential, MPI [78] or PVM [80] jobs), executing these workflows in Globus-based computational Grids relying on user credentials, and finally, analysing the monitored trace-data by the built-in visualization facilities. This portal provides the following functions: defining Grid environments, creation and modification of workflow applications, managing Grid certificates, controlling the execution of workflow applications on Grid resources and monitoring and visualizing the progress of workflows and their component jobs.

The P-GRADE Portal is interfacing several Grid brokers to reach the resources of different Grids in an automated way. As workflow managers schedule the actual job submissions in portals, they should be set to utilize brokers. In this portal Condor DAGMan [82] is responsible for workflow execution. Although it itself cannot invoke Grid services, it supports customized Grid service invocations by its pre/scheduler/-post script concept [82]. One pre and one post script can be attached to every node (job) of a DAGMan workflow. DAGMan guarantees that it first executes the pre script, then the scheduler script and finally the post script when it reaches a workflow node. Consequently, the Portal Server automatically generates appropriate pre, scheduler and post scripts for every workflow node when the workflow is saved on the server. These scripts can handle GridFTP transfers and submit jobs to GRAM clients of the connected Grids. These scripts are used in the same way in both single- and multi-grid configurations (multi-grid means jobs are set to different Grids, and setting a job to a Grid means that among the node properties some Grid or its broker is selected from a drop-down list), however the contents of these portal scripts depend on the actual Grid (different Grids have different commands). In general, when DAGMan processes a node (job) in a workflow that is set to a broker, first it invokes the pre script that prepares the broker utilization (e.g. copy remote input files), the

scheduler script submits the job to the broker, and the post script tracks job states until the execution is finished. The broker provides information about the actual job status and the post script notifies the portal about the status changes.

Currently the portal can utilize GTbroker for Globus 2, 3 [99] and LCG-2 Grids [92], the WMS of the LCG-2/gLite middleware and the broker of NorduGrid middleware [122]. The jobs of the workflow that require EGEE services can run on an EGEE type of Grid [92]; jobs that require only Globus services can be mapped to resources handled by GTbroker, and finally the NorduGrid Broker can be utilized to run jobs on resources of the NorduGrid ARC. Different Resource Brokers usually require different user job descriptions. In the Workflow Editor of the portal the users can choose a broker for each job of the defined workflow. According to this setting a Language Editor pops up, where the user can edit the attributes through the GUI fields. This Editor generates an RSL, JDL or an xRSL [122] file from these job requirements, depending on the middleware of the target Grid. The scheduler script of DAGMan invokes the brokers with these descriptions. In case of Globus-based Grids the file movements are also handled by GTbroker, so the scheduler script only needs to activate and run an instance of GTbroker. In case of EGEE WMS and NorduGrid broker, there are special commands for tracking job states and retrieving the output and log files, therefore the scheduler script needs to call these services, too. In case of remote files only the EGEE brokers use a so-called "close to file" policy: it means they try to place the job to a resource, which is the nearest to the location of its remote input files. Since most of these brokers do not handle remote file transfers, the portal workflow manager solves this problem again based on DAGMan services. DAGMan submits a wrapper script as the executable, carrying all the job files and descriptions. After this script is started on the selected computing resource, it handles the remote input file transfers and – after the real job execution – the remote output file transfers between the storage elements and the actual computing element. With this solution all kinds of file transfers can be carried out during broker utilization. This solution was described in detail for supporting MPI applications in [27]. The portal has a certificate manager portlet, which is responsible for managing X.509 certificates. Users can upload certificates into MyProxy servers and download GSI proxies that are handled automatically by the workflow manager (proxies are mapped to VOs).

Figure 2.7 shows how multiple broker utilization is carried out in the P-GRADE Portal. In this way a workflow can be brokered over several Grids based on different

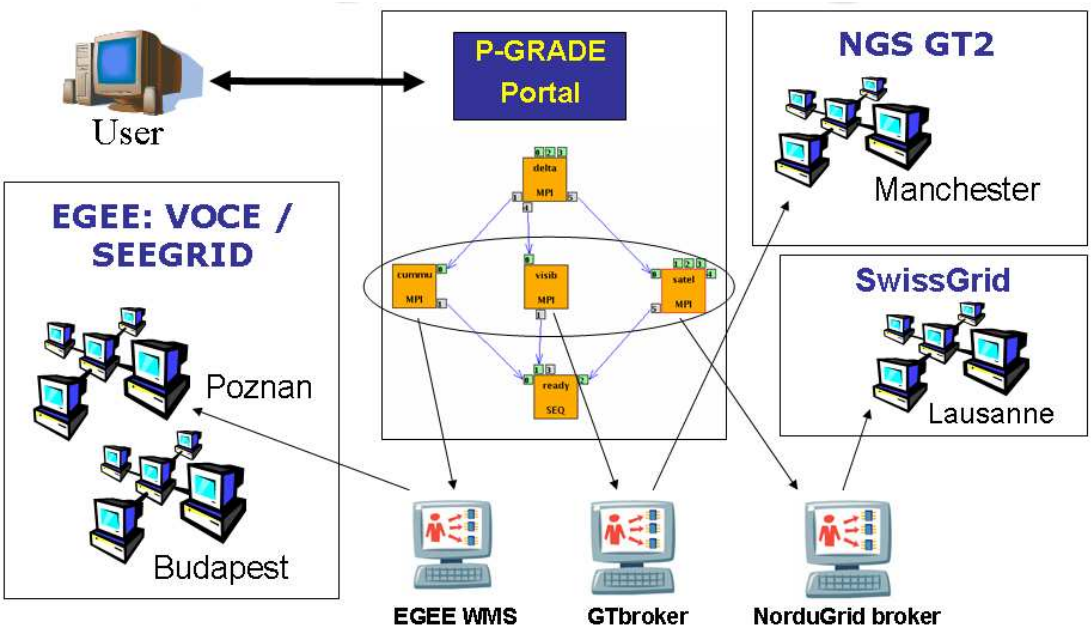

Figure 2.7: Multiple-broker utilization in the P-GRADE Portal.

underlying Grid technologies and still providing the most beneficial utilization of the available resources. A workflow, depicted in the figure, is a directed acyclic graph that connects sequential or parallel programs into an interoperating set of jobs. The nodes of such a graph are batch jobs, while the arc connections define data relations among these jobs. Arcs define the execution order of the jobs and the input/output dependencies that must be resolved by the workflow manager during execution. After a user has defined a workflow and has set the jobs to brokers, the execution can be started. The three jobs in the middle of the workflow do not depend on each other, therefore they can be started simultaneously. Since they are set to different VOs and brokers, multi-brokering is performed: the first job is submitted to an EGEE VO through the WMS, the second through GTbroker to NGS and the third through the NorduGrid broker to the Swiss Grid.

The usage of Grid portals performing multiple-broker utilization contributes to solve the Grid Interoperability issue. Furthermore this interoperable workflow execution means higher performance and provides shorter makespan most of the time. Let us imagine a simple scenario, when we set all jobs of a workflow to one middleware (that usually means we submit the jobs to the same broker), and set a similar workflow to utilize more middleware systems as shown in Figure 2.7. It is easy to see that the second workflow instance can access more computing power. The more load the

middleware of this first workflow has, the more time the second workflow can save (even if it also uses the same middleware for most of the jobs). More information on interoperability at workflow level with the P-GRADE portal can be seen in [45]. The contribution representing the multi-grid brokering from the P-GRADE Portal is a joint result. My own contribution includes the porting of GTbroker into the portal and the scenario that describes multi-brokering.

Even though these instant, short-term approaches *can establish interoperation* among different Grids, there are certain *disadvantages* that should be *eliminated*. Extending an existing broker with support for more and more Grids can result in a much-too-robust system, and most broker components would need modifications. Interfacing brokers to portals for additional Grid support does not need the redesign of the brokers, but still some portal components have to be modified. Though current portals provide a transparent access to various Grid services, an additional disadvantage of this approach is that users need to know broker capabilities, and manually select from the available brokers. Users could learn the capabilities of the utilized brokers, but they are *lacking* dynamic information, such as successful submission rate, background resource load of the brokers, reliability of the brokers and so on. A deeper investigation of Grid resource management is needed in order to find a place, where a sustainable, well-designed, high-level brokering service could be created eliminating these problems.

2.6 Summary

In this chapter I introduced Grid brokering by presenting a survey that gathers the relevant tools currently available in the literature for user communities to access resources of different Service Grid systems. I analyzed these brokers and developed a taxonomy that categorizes broker properties. I gathered definitions for Grid Interoperability that together with the taxonomy represent the problem area of this dissertation. I also demonstrated how multi-grid brokering can be achieved by brokers and portals as a short-term interoperable solution. The results of this chapter belong to thesis I, and were published in paper [P1], [P2], [P4], [P5], [P11]. In the next chapter I make a closer connection between brokering and interoperability and present an informal and formal model for interoperable Grid brokering.

Chapter 3

Analysing and modeling Grid brokering

In order to solve interoperability with Grid resource management, we need to have a deeper insight to this area. In the previous section I have introduced Grid brokering and established a Grid resource brokering taxonomy to determine what properties brokers possess and what functionalities are desired for certain tasks. The presented survey shows that the currently available Grid resource management tools are built on different middleware components supporting different properties and named with a bunch of acronyms – even the ones having similar purposes. This plethora of approaches formed the domain of Grid resource management into a grey box with blurry boundaries where neither the users nor the researchers can clearly see how these tools are related and what their relevant properties are. Until the definitions and interrelations are clarified, further development and interoperability cannot be facilitated. Therefore, in the next section I analyse the resource management layer and give informal definitions of brokering entities revealed by a Grid resource management anatomy. Further on I will investigate how brokering methods of Grid resource management can be formalized and what essential functionalities related to interoperability can be separated based on a formal model.

3.1 The anatomy of Grid resource management

Taking a closer look on current Grid resource managers and research directions we are facing confusing *contradictory* definitions and references. When searching and comparing related works we meet meta-schedulers, local and global schedulers, brokers, resource allocation managers and some other related expressions. In this section I

gather and classify these expressions used in the area of Grid Resource Management regardless of the different types of Grids they might be connected to, or used in. I refer to these definitions further on in this dissertation by naming specific components, building blocks of these solutions. Similarly to the pioneer work preformed by I. Foster et. al. [33] to map the relevant actors of Grid systems, I have created the *anatomy of Grid resource management*. As a result of my analysis, we can see my view on the architecture of a Grid system focusing on resource utilization in Figure 3.1. All the abbreviations used in this figure denote collections (groups) of *similar* components or services. The arrows denote possible connections between these groups of components, and not all these components are needed for successful job submissions: some users prefer to use portals for application development and execution, and others may access lower-level resource managers directly. In the following I introduce these groups by gathering the generally used acronyms and expressions for components having similar roles and utilization scopes.

R – resource: In general it means a physical machine, where user programs are executed. We can distinguish between three types regarding their utilization: Computing Element (CE), where the user program (also task or job) execution takes place, Storage Element (SE), where user data is stored, and Instrument Element (IE) as defined in the domain of remote instrumentation. Remote Instruments (RI or IE as Instrument Element [34]) are various tools to access, process, produce or consume data usually by physicists, chemists, biologists or other researchers. As a higher level abstraction of these components, Web Services (WS) or Grid Services (GS) could also be regarded as resources, since they similarly produce output for a given user input.

LRMS – local resource management system, scheduler, local scheduler, local resource manager, sub-scheduler: These tools are usually cluster or queue managers that were taken from high-performance and distributed computing, and now generally used in Grid Systems. The widely used examples are PBS, SGE and LSF [26].

RAM – resource access manager: This is usually a component of a Grid middleware that accesses the resource directly and handles the arriving Grid jobs. It provides an interface for the middleware to the resource. An example is GRAM in the Globus Toolkit [99].

RMS – Grid resource management system, global scheduler, meta-scheduler, resource broker, Grid scheduler, orchestrator: They usually interact with one or more components of a Grid middleware. An RMS can be an

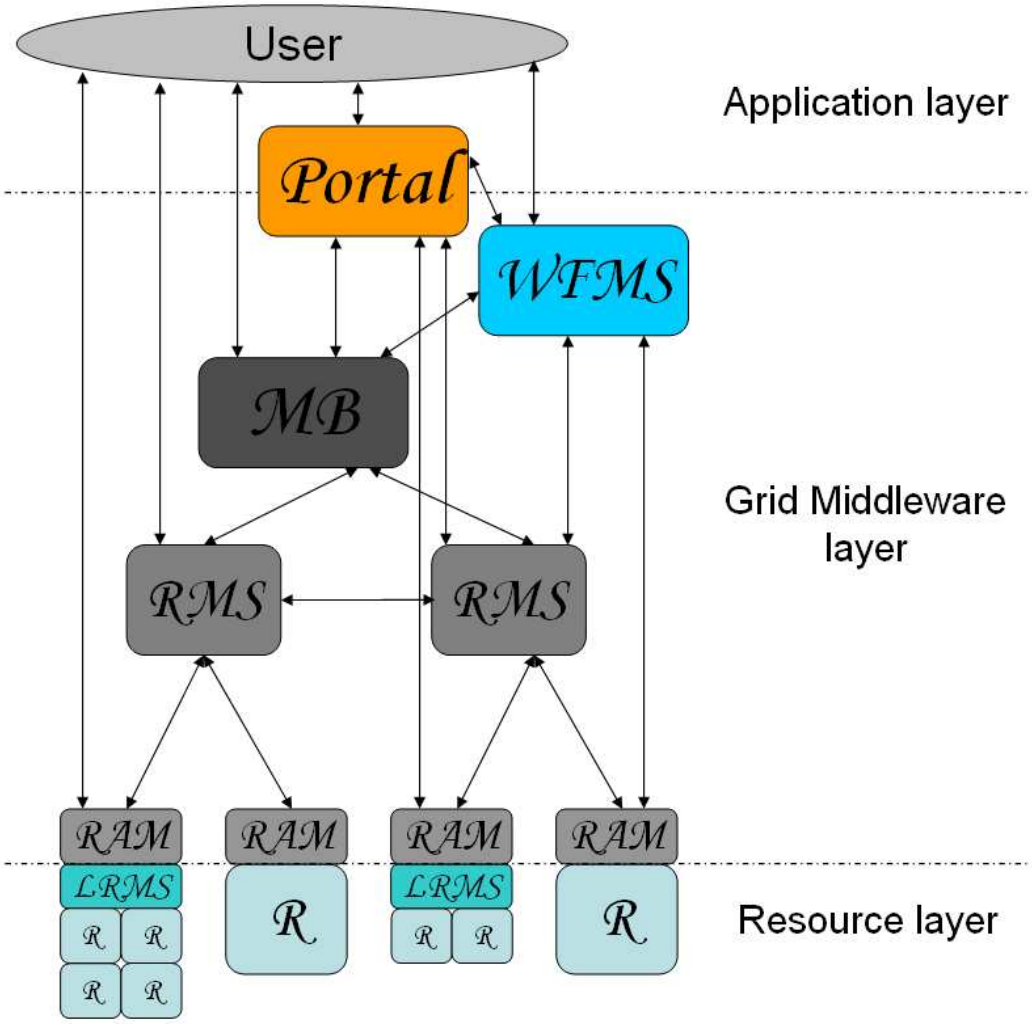

Figure 3.1: Grid resource managers and their connections

internal middleware service, or an external tool that uses other middleware components or services. Its role is similar to the role of LRMS – this is the reason why they can be confused sometimes. While the LRMS usually schedules user tasks within a Grid resource (e.g. a cluster) among free processors, the RMS schedules jobs at the Grid middleware layer by selecting a resource (R) that best matches the requirements of the user jobs. (Nevertheless the selected resource can also be a cluster with an LRMS.) Therefore an LRMS is generally called a scheduler, and the RMS is a meta-scheduler. The other listed expressions for RMS basically mean tools used for the same purpose (i.e. selecting an R for a job). Some of them only slightly differ, some of them support different middleware solutions, job types, agreements or various

quality of services. The taxonomy (in Section 2.2) introduces these properties and the differences among the currently used brokers belonging to this group.

MB – meta-broker, inter-domain broker, intergrid gateway: Meta-brokering is a novel approach that introduces another level above current Grid resource managers in order to facilitate inter-Grid load balancing. I target this part of the resource management layer in order to solve interoperability problems. This approach is introduced and discussed in detail in Section 4.1 of the next chapter. The Grid meta-broker can be placed on top of the resource brokers, it uses meta-data to decide where to send a user's job. Another approach for meta-brokering was introduced in [40] and [72], in which similar broker instances communicate with each other in a peer-to-peer fashion and exchange jobs, when the locally managed domains are saturated.

WFMS – workflow management system, workflow enactor, workflow manager, workflow scheduler: The role of this component is to execute complex user applications called as workflows. Its core component is a workflow scheduler, which is responsible for submitting the jobs in the workflow in a way that the execution of the whole workflow will be the cheapest regarding some cost function (time, budget, etc.). Workflow enactors or managers [88] can be regarded as complex systems having more components (which I do not detail here). These workflow enactors are usually connected to one or more RMSs that take care of the actual job scheduling. The workflow management system can also be connected to more middleware services. This tool may incorporate resource-related information into its scheduling (not only workflow properties), in this case it is better to access the resources directly and neglect the usage of other Grid resource managers. I mention this category only for the sake of completeness, and do not detail it further in this dissertation.

Portal – Grid portal, problem solving environment: This tool provides a high-level user friendly environment for accessing Grid services. Members of this group generally use graphical user interfaces to define jobs, create workflows, submit them and track their states. They are usually connected to, or incorporate some WFMS or RMS to manage Grid resources. More information on Grid portals can be found in [46].

Further on in this dissertation I focus on two groups of this architecture: the RMS and the MB. As a *second view* of the anatomy, Figure 3.2 is used to reveal some small differences among the used expressions within these groups. In general a meta-scheduler is focusing more on job scheduling (executing an algorithm), a re-

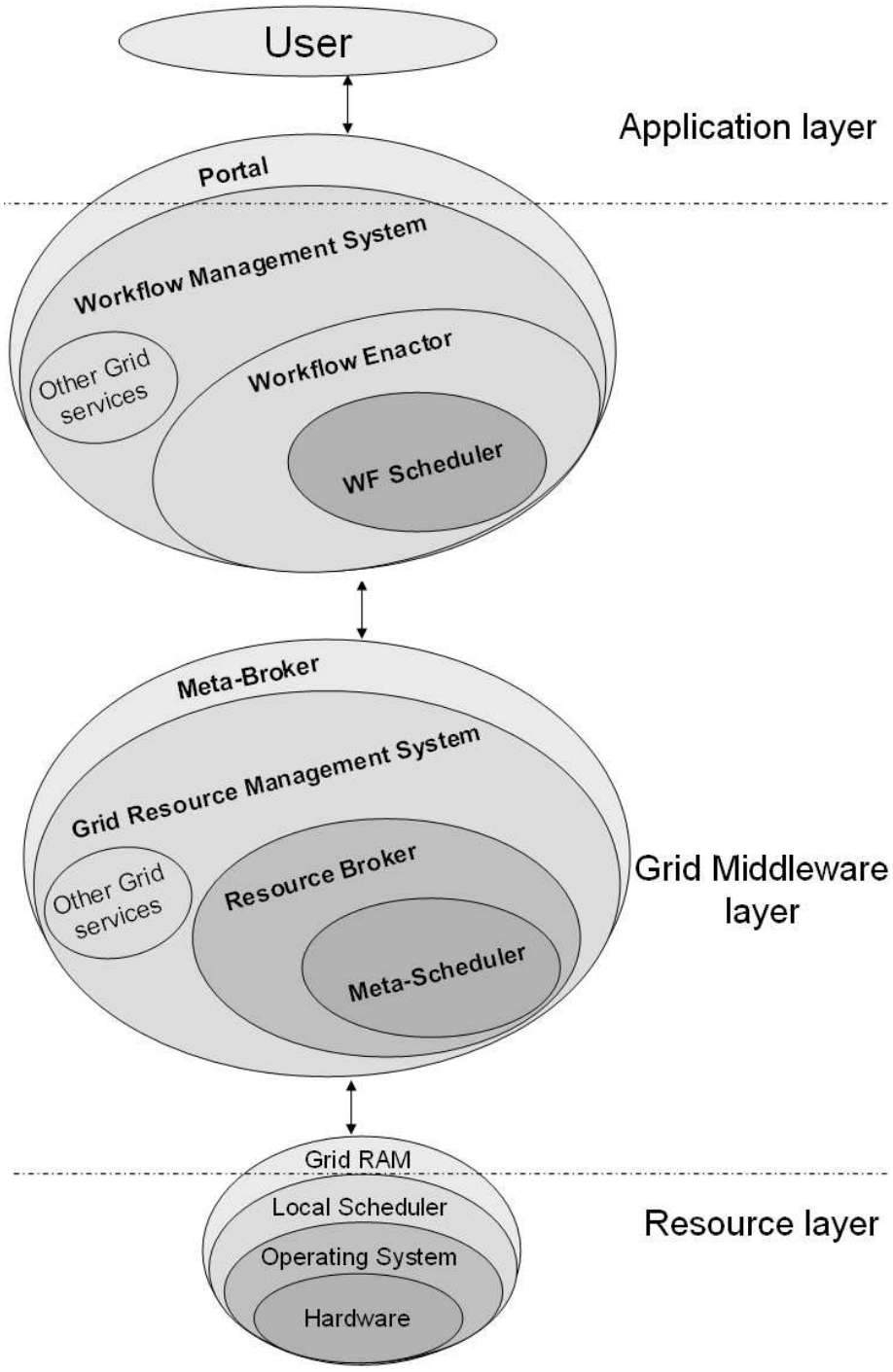

Figure 3.2: The anatomy of Grid resource management

source broker incorporates more services such as job handling and job state tracking, while a resource management system takes care of the whole job management process with job accounting and other related services. A meta-scheduler and a broker can communicate with and use other services of the middleware to be able to manage the whole job's life-cycle. In this sense scheduling is an atomic process, brokering incorporates scheduling, and resource management includes brokering.

3.2 An extended formal model for Grids

In the previous anatomy I have shown *informally* how resource management is carried out by various tools from the Resource layer up to the Application layer of Grid systems. The aim of this section is to establish a *formal*, semantic model for Grid resource management using Abstract State Machines (ASM) based on the anatomy. In order to formalize the *different brokering levels* shown in the anatomy in Section 3.1, I refer and *extend* the formal definition of Grid Computing published by Zs. Németh, and V. Sunderam in 2003 [60]. That time there had been several definitions for Grid Computing without the ability of making a clear distinction between Grids and other distributed systems. This work concluded that Grids cannot be defined purely by their properties, rather their runtime semantics make the real difference. Based on the analysis, a formal definition was given for Grid Computing revealing its essential and characteristic functionalities. The aim and methodology of my contribution regarding Grid brokering is similar: establishing a formal, semantic model for Grid resource management using Abstract State Machines (ASM) that clarifies the relations of different brokering approaches. I extend the formal model for Grids by classifying brokering components into three categories and defining three agents for resource management at different levels of the Grid middleware.

First I give a brief introduction of the formal Abstract State Machine (ASM) method. ASM represents a mathematically well founded framework for system design and analysis [6]. It is able not just to model a working mechanism precisely but also to reveal the highly abstract nature of a system, therefore it can easily be tailored to the required level of abstraction. Logician structures applied in ASMs offer an expressive, flexible and complete way of state description. The basic sets and the functions interpreted on them can be freely chosen to the required level of complexity and precision.

In ASM, a signature (or vocabulary) is a finite set of function names, each of fixed

arity. Furthermore, it also contains the symbols $true, false, undef, =$ and the usual boolean operators. A state A of signature Υ is a non-empty set X together with interpretations of function names in Υ on X. X is called the superuniverse of A. An r-ary function name is interpreted as a function from X^r to X, a basic function of A. A 0-ary function name is interpreted as an element of X [108]. A location of A (can be seen like the address of a memory cell) is a pair $l = (f, a)$, where f is a function name of arity r in vocabulary Υ and a is an r-tuple of elements of X. The element $f(a)$ is the content of location l. An update is a pair $a = (l, b)$, where l is a location and b is an element of X. Firing a at state A means putting b into the location l while other locations remain intact. The resulting state is the sequel of A. It means that the interpretation of a function f at argument a has been modified resulting in a new state. ASMs are defined as a set of rules. An update rule $f(a) := b$ causes an update $((f, a), b)$, i.e. hence the interpretation of function f on argument a will result b. It must be emphasized that both a and b are evaluated in A. The nullary $Self$ function allows an agent to identify itself among other agents. It is interpreted differently by different agents (that is why it is not a member of the vocabulary.) An agent a interprets $Self$ as a while an other agent cannot interpret it as a. The $Self$ function cannot be the subject of updates. A conditional rule R is of form

if c
  then R_1
else
  R_2
endif

To fire R the guard c must be examined first and whenever it is true R_1, otherwise R_2 must be fired. A block of rules is a rule and can be fired simultaneously if they are mutually consistent. Some applications may require additional space during their run, therefore the *reserve* of a state is the (infinite) source where new elements can be imported from by the following construct

extend U **by** $v_1, \ldots, v_n$ **with**
  R
endextend

meaning that new elements are imported from the *reserve* and they are assigned to universe U and then rule R is fired [108].

The basic sequential ASM model can be extended in various ways like non-deterministic sequential models with the choice construct, first-order guard expressions, one-agent parallel and multi-agent distributed models. A distributed ASM [6] consists of a finite set of single-agent programs Π_n called modules, a signature Υ, which includes each Fun(Π_n)-$\{Self\}$, i.e. it contains all the function names of each module but not the nullary $Self$ function, and a collection of initial states.

As it can be seen, ASM states are represented as (modified) logician's structures, i.e. basic sets (universes) with functions interpreted on them. Structures are modified in ASM to enable state transitions for modeling dynamic systems. Applying a step of ASM M to state (structure) A will produce another state A' on the same set of function names. If the function names and arities are fixed, the only way of transforming a structure is to change the value of some functions for some arguments. Therefore, the most general structure transformation (ASM rule) is a guarded destructive assignment to functions at given arguments [6].

Refinement [6] is defined as a procedure where abstract and more concrete ASMs are related according to the hierarchical system design. At higher levels of abstraction implementation details have less importance whereas they become dominant as the level of abstraction is lowered giving rise to practical issues. Its goal is to find a controlled transition among design levels.

I am not aware of any other works that investigate formal models specifically for Grid resource manager components. Bratosin et al. proposed a reference model for Grid architectures based on coloured Petri nets in [8]. Though they provide a definition for job scheduling, they do not detail brokering steps and mechanisms at different levels. Altenhofen et al. investigated Service Oriented Architectures in [3], more specifically service discovery, mediation and composition. These components have some similar functionalities but this work is more focused on a unified, higher level service framework, and do not explore resource manager components. Börger et al. proposed an ASM model for workflows in [7]. The work presents workflow interpretations and transitions, which are related to this model, but they stay at the Application layer and do not deal with brokering at job level whereas my model targets the middleware below the Application layer.

Before I define the model, I present the ASM for Grids defined in [60] with some modifications. Figure 3.3 shows the important elements of the initial model. The

ASM universes of the model are depicted on the left of the figure, and on the right a graphical representation of the connections of some elements of these universes and the most relevant functions governing process execution are shown. In this model user applications consist of one or more processes (denoted by p in the figure), while Grids consist of several nodes (n) having one or more resources (r). During the execution of the user application first an agent maps the actual process of the application to a resource in the Grid, then the process is installed on the node of the resource as a task (t), which starts to use the resource. When all the processes of the application finished using their resources, the application is finished.

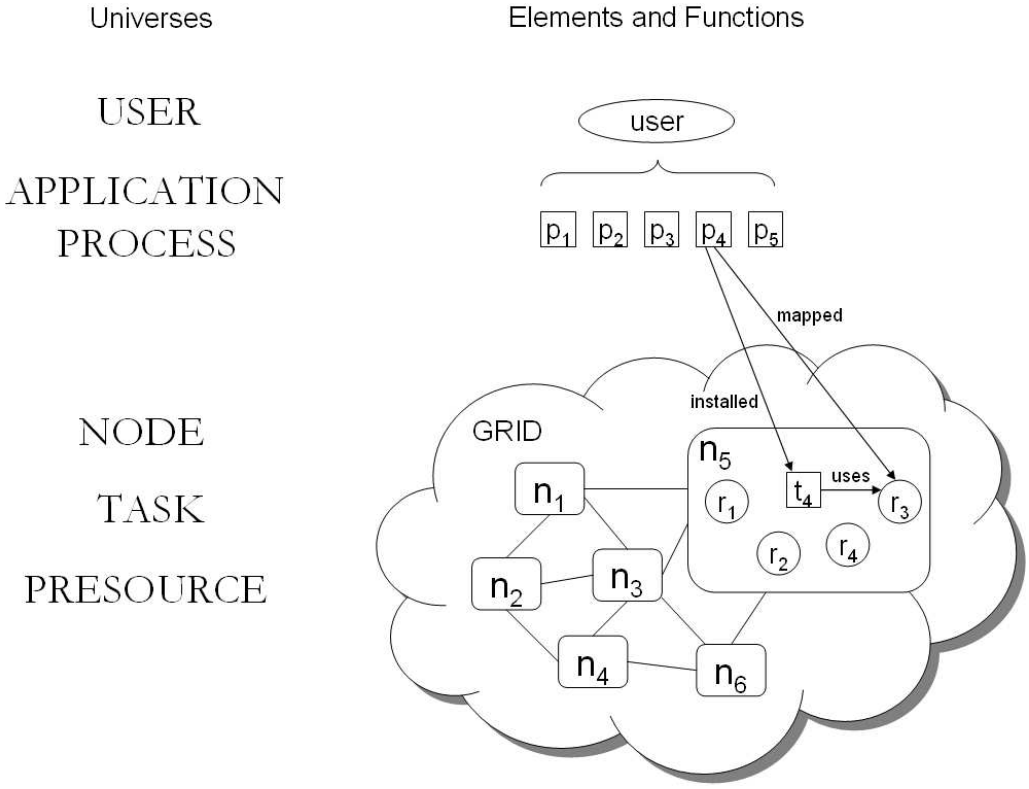

Figure 3.3: Basic elements of the initial ASM model for Grids.

I extend this *formal* model by introducing *Grid brokering* at different levels. I reuse the initial model of Grid systems introduced in [60] in a slightly modified form here. The modification is indicated by introducing more practical issues related to realization; by aligning the model to the terminology and naming conventions of Grid brokering; and finally by experiences in Grid Computing since the paper was published. These modifications do not invalidate or alter the content and conclusion of the initial model just add more relevant details. The modifications are shown in

Figure 3.4, in which I emphasise that a user application consists of jobs (j) that can have one or more processes (p). A Grid consists of hosts (h) that have one or more resources (r).

In the following subsections I define the basic elements of my extended formal model based on ASM: the universes, the signature and the rules.

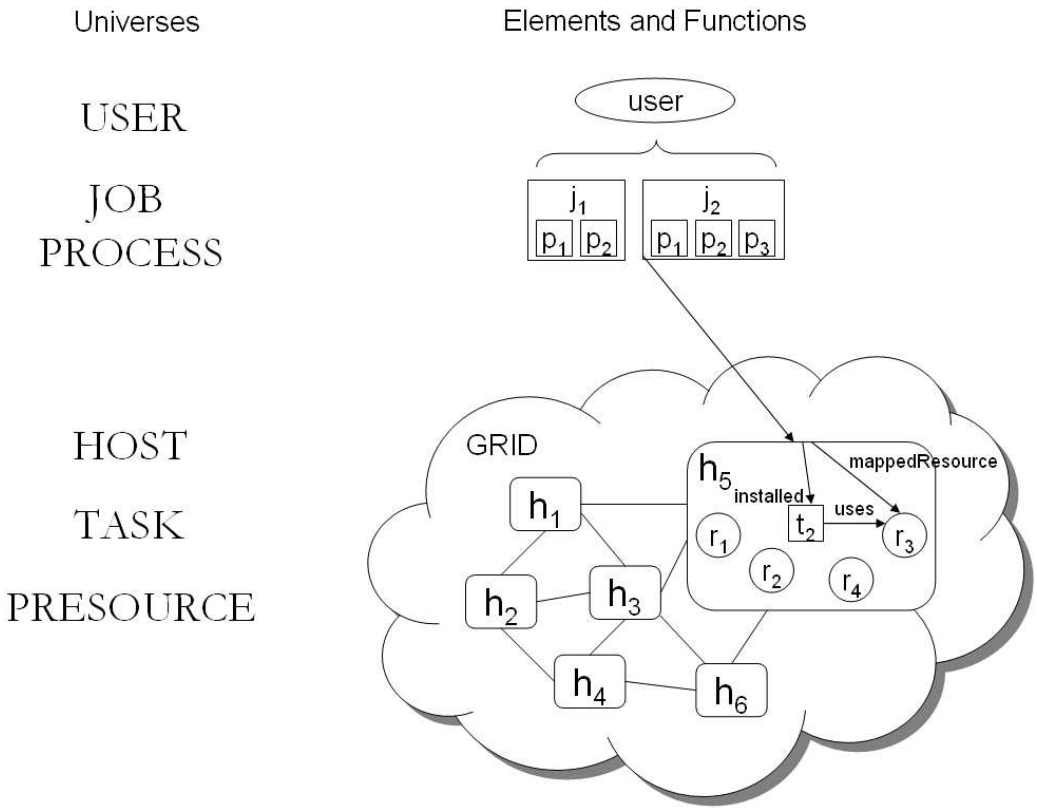

Figure 3.4: The modified ASM model for Grids.

3.2.1 Universes and signature

To define this formal framework, first I need to examine real service Grid systems. Certain objects of the physical reality are modeled as elements of universes and relationships between real objects are represented as functions and relations. In Grid systems users (universe $USER$) define their applications in the form of jobs (universe JOB), which is the most typical computing paradigm for Grids, hence I restrict my model to this case. A job consists of one or more processes (universe $PROCESS$). The installed instances of processes are called tasks (universe $TASK$), which can

be run on different hosts (universe $HOST$). Hosts are the building blocks of Grid systems, and typically a job is mapped, then sent to a host for execution. A host may have several nodes (e.g. when a host is a cluster), and nodes have certain resources that processes require to run. Since nodes are usually invisible (and unmanageable) for higher level tools, therefore I neglect them in this model. In this way one or more physical resources (universe $PRESOURCE$) belong to a host, which also determines the physical location (universe $LOCATION$) of the resources. The processes of jobs require some of these resources to run. Users should select a host according to these resource requirements, which are called as abstract resources (universe $ARESOURCE$). Information on the physical resources of the hosts can be gathered by querying the information system of a Grid.

Once a job is *submitted* to a host, it is mapped to physical resources during execution. While a resource is busy, the mapped process is in *waiting* state. When the resource becomes free, the process starts using it and enters *running* state. Process termination implies a *done* state in case of successful run, and a *failed* state in case of an error. In general, Grid authorization allows users to log in to some hosts and validates user privileges to use some resources of some hosts [32]. The requested (abstract) and the physical resources have certain attributes (universe $ATTR$). Compatibility between an abstract and a physical resource means the physical resource can satisfy the process requirement. According to this informal description, the following functions are used in the model:

job: $PROCESS \rightarrow JOB$
user, globaluser, localuser: $JOB \rightarrow USER$
submitted: $JOB \times HOST \rightarrow \{true, false\}$
procRequest: $PROCESS \times ARESOURCE \rightarrow \{true, false\}$
uses: $PROCESS \times PRESOURCE \rightarrow \{true, false\}$
mapped: $PROCESS \rightarrow LOCATION$
belongsTo: $PRESOURCE \times HOST \rightarrow \{true, false\}$
installed: $TASK \times LOCATION \rightarrow \{true, false\}$
attr: $\{ARESOURCE, PRESOURCE\} \rightarrow ATTR$
location: $PRESOURCE \rightarrow LOCATION$
handler: $PRESOURCE \rightarrow PROCESS$
type: $PRESOURCE \rightarrow ATTR$
compatible: $ATTR \times ATTR \rightarrow \{true, false\}$

canLogin: $USER \times HOST \rightarrow \{true, false\}$
canUse: $USER \times PRESOURCE \rightarrow \{true, false\}$
jobState: $JOB \rightarrow \{submitted, waiting, running, done, failed\}$
procState: $PROCESS \rightarrow \{waiting, running\}$
event: $TASK \rightarrow \{start, abort, terminate\}$
mappedHost: $JOB \rightarrow HOST$
mappedResource: $PROCESS \times ARESOURCE \rightarrow PRESOURCE$

3.2.2 Initial state

I assume that k processes belong to a job of a user. The job and its processes have some requirements, and no process and job is mapped to any resource or host. Therefore the states of the jobs and processes are undefined. In the following I define the initial state of my model:

$\exists p_1, p_2, \ldots, p_k \in PROCESS : job(p_i) \neq undef, 1 \leq i \leq k$
$\forall p_i, 1 \leq i \leq k : \text{user}(p_i) = u \in USER$
$\forall p_i, 1 \leq i \leq k : \exists ar \in ARESOURCE : \text{procRequest}(p_i, ar) = true$
$\forall p_i, 1 \leq i \leq k : \exists pr \in PRESOURCE : \text{uses}(p_i, pr) = false$
$\forall j : \text{mappedHost}(j) = undef$
$\forall p_i, 1 \leq i \leq k : \text{task}(p_i) = undef$
$\forall p_i, 1 \leq i \leq k : \text{mapped}(p_i) = undef$
$\forall j : \text{jobState}(j) = undef$
$\forall p_i, 1 \leq i \leq k : \text{procState}(p_i) = undef$
$\forall u \in USER, \exists pr_1, pr_2, \ldots, pr_m \in PRESOURCE :$
$\text{canUse}(u, pr_i) = true, 1 \leq i \leq m$

After I have defined the universes and the signature, in the following I give the rules of my model that constitute a module, i.e. a program that is executed by each agent in the model. The model presented here is a distributed multi-agent ASM where agents are jobs, i.e. elements from the JOB universe. The working behaviour of the brokering model is depicted from the perspective of the jobs, hence the self function is represented as j and means the identity of a job, i.e. it can identify itself among other agents. It is interpreted differently by different agents.

3.2.3 Rule 1: Resource selection

According to Figure 3.4, when the job is sent to a host, the required resources need to be selected that are used by the processes of the job. During job execution, a task of each process is installed to the location of the required and selected resource. The precondition of resource selection is that the process of the job should be able to use the mapped resource. In case of the process can directly access the physical resource (r_d) the execution (resource usage) is automatically started, otherwise a local handler process should provide the execution platform (i.e. the additional software or service). If this handler process does not exist, it should be started before execution. The agent responsible for resource mapping needs to ensure that the chosen resource fulfils the abstract resource requirement of the process. Here is the formal definition:

let h = mappedHost(j)
let job(p) = j
let pr = mappedResource(p, ar)
if ($\exists ar \in ARESOURCE$):
  procRequest(p, ar) = $true$ & $pr \neq undef$ & canUse(user(p), pr) = $true$
    then if type(pr) = r_d
      then mapped(p) := location(pr)
      installed(task(p), location(pr)) := $true$
    else if ($\neg\exists p' \in PROCESS$): handler(pr) = p'
        extend $PROCESS$ **by** p'
        with mapped(p') := location(pr)
        installed(task(p'), location(pr)) := $true$
        handler(pr) := p'
        do forall $ar \in ARESOURCE$
          procRequest(p', ar) := $false$
        enddo
      endextend
    endif
    procRequest(p, ar) := false
    uses(p, pr) := true
endif

$\Pi_{\text{resource\_mapping}}$

if ($\exists ar \in ARESOURCE, \exists p \in PROCESS, \exists h \in HOST$):
job(p) = j & mappedResource(p, ar) = $undef$ &
procRequest(p, ar) = $true$ & h = mappedHost(j)
  then choose pr **in** $PRESOURCE$
    satisfying compatible(attr(ar), attr(pr)) & belongsTo(pr, h) = $true$
      mappedresource(p, ar) := pr
  endchoose
endif

Here, I note that though generally a job runs on a host (if it is a parallel job of communicating processes, it runs on a number of resources of this host in parallel), some middleware tools may enable co-allocation of parallel processes on nodes of different hosts. I do not deal with this situation, since it is rarely used and supported, but further refinement of this model could represent such cases.

Before job execution it is necessary to authenticate users. In Service Grids users are authenticated by proxies of Grid certificates [32]. A local process is responsible for validating these proxies by mapping global users to local ones having the same privileges. The related formalism of user mapping is similar to the one presented in [60]:

$\Pi_{\text{user\_mapping}}$

let pr = mappedresource(p, ar)
if ($\exists ar \in ARESOURCE, \exists p \in PROCESS$):
  procRequest(p, ar) = $true$ & $pr \neq undef$ & canUse(user(p), pr) = $true$
    then if type(pr) = r_d | ($\exists p' \in PROCESS$) : handler(pr) = p'
      then choose u **in** $USER$
        satisfying canLogin(u, location(pr))
          usermapping(globaluser(p), pr) := u
      endchoose
    else if ($\exists p' \in PROCESS$): handler(pr) = p'

then usermapping(globaluser(p), pr) := localuser(handler(pr))
endif
endif

3.2.4 Rule 2: State transition

In this subsection I define, how job states are evolving during execution:

if ($\exists h \in HOST$): submitted(j, h) = $true$
then jobState(j) := $submitted$
endif
if ($\exists p \in PROCESS$): job(p) = j & mapped(p) $\neq$ $undef$
then procState(p) := $waiting$
jobState(j) := $waiting$
endif
if ($\exists pr \in PRESOURCE, \exists p \in PROCESS$): job(p) = j & uses(p, pr) = $true$
then procState(p) := $running$
jobState(j) := $running$
endif
if ($\exists p \in PROCESS, \exists t \in TASK, \exists pr \in PRESOURCE, \exists h \in HOST$):
uses(p, pr) = $true$ & $belongsTo(pr, h) = true$
& installed(t, h) = $true$ & event(t) = $abort$
then jobState(j) := $failed$
$PROCESS(p) := false$
endif

Though in general, process spawning could cause additional resource requests for job execution in a host, I do not detail this in my model, and keep it as abstract as possible, since at the level of Grid brokering process communications and spawning are invisible. In order to handle these situations, I assume that resource requests of spawned processes are known a priori. State transitions related to job termination are formalized in Rule 3.

3.2.5 Rule 3: Termination

Job execution is terminated under the following conditions:

if ($\exists p \in PROCESS$):
  job(p) = j & procState(p) = $running$ & event(task(p)) = $terminate$
    then $PROCESS(p) := false$
endif
if ($\exists p_1, \ldots, p_m \in PROCESS$): job(p_i) = j & jobState(j) = $failed$, $1 \leq i \leq m$
  then $PROCESS(p_i) := false$
endif
if ($\neg\exists p \in PROCESS$): job(p) = j & jobState(j) = $running$
  then jobState(j) := $done$
endif

3.3 ASM model for Grid brokering

Now that we have the revised formal model for Grids, in this section I focus on middleware components responsible for *brokering* in Grids. In this ASM model these components are represented by agents. First I recall the related parts of the informal anatomy introduced in Section 3.1 then show how these resource manager components can appear as agents in the formal model described above. Furthermore I emphasize how these brokering components contribute to Grid Interoperability, i.e. how they may support transparent job submissions to different, separated Grids.

At the lowest level of Grid resource management we can find local resource managers (LRMS) that were taken from high-performance and distributed computing, and now generally used in Grid Systems. This local resource management is formalized in Rule 1 of this model. Without additional brokering components users need to choose from the available hosts manually relying on mostly static information. One level above, Grid resource managers (RMS), also called Grid brokers, are needed to automate host selection. With the help of Grid brokers, host selection is automated, but users are still bounded to separate Grids (i.e. Grid systems that are complete systems on their own but closed to any form of interoperability between each other, either by technology, compatibility, administrative or other restrictions) managed by

their own brokers. Nevertheless users have the ability to select manually, which broker and Grid they would like to use (even static information on broker properties are available in form of manuals or taxonomies). In order to achieve better interoperability broker selection should also be automated. Therefore we need a novel approach called *meta-brokering*, which introduces another layer above current Grid brokers in order to facilitate inter-Grid load balancing and interoperable brokering.

A typical Grid usage scenario for a *job execution* that requires the following steps:

1. The user defines its application as jobs, also stating the requirements of its execution.

2. The user requirements of the job is examined by the meta-broker, and mapped to the properties of the available brokers. A proper broker, that is able to submit the job, is selected for submission.

3. The selected broker examines the resource requirements of the job and matches them to the physical resources of the available hosts. A host having all the required resources is selected for execution.

4. The agent on the selected host (the local resource manager) maps the resource requirements of the job to the available physical resources during execution.

In the following subsections I define two more rules to model the informal description and discussion above. We need additional universes and functions to describe brokering functionalities in this model.

3.3.1 Rule 4: Host selection for Grid brokering

Brokers (universe $BROKER$) are responsible for host selection, therefore hosts are managed by brokers, which can have different properties (universe $PRORERTY$) that users may require for job execution. A user should select a broker for its job according to these requirements (universe $REQUIREMENT$). Furthermore I place universe $ARESOURCE$ as a subset of universe $REQUIREMENT$, since the elements of both sets represent user requirements, and universe $PRESOURCE$ can be a subset of universe $PRORERTY$, because physical resources can be regarded as host properties. The following functions are added to the model:

request: $JOB \times REQUIREMENT \rightarrow \{true, false\}$
submitted: $JOB \times \{HOST, BROKER\} \rightarrow \{true, false\}$
manages: $BROKER \times HOST \rightarrow \{true, false\}$
have: $BROKER \times PRORERTY \rightarrow \{true, false\}$
attr: $\{REQUIREMENT, PRORERTY\} \rightarrow ATTR$

I extend the initial state by:

$\forall j : \exists r \in REQUIREMENT :$ request$(j, r) = true$

I extend Rule 2 with the following state changes:

if ($\exists b \in BROKER$): submitted(j, b) $= true$
  then jobState(j) := *submitted*
endif
if ($\exists h \in HOST$): submitted(j, h) $= true$
  then jobState(j) := *waiting*
endif

Once a broker is selected by the user, it should find an execution host. The precondition of this host selection process is that the user of the job should be able to use the required resources of the selected host. The broker agent responsible for host mapping needs to ensure that the chosen host has all the resources requested by the processes of the job. This additional component responsible for Grid brokering is highlighted in Figure 3.5. In the following I state the formal definition for Rule 4 that performs host selection:

$h =$ mappedHost(j)
if ($\exists ar_1, \ldots, ar_m \in ARESOURCE, \exists pr_1, \ldots, pr_m \in PRESOURCE$):
  request(j, ar_i) $= true$ & $h \neq undef$
  & canUse(user(j), pr_k) $= true$, belongsTo(pr_k, h) $= true, 1 \leq i, k \leq m$
    then submitted(j, h) := $true$
endif

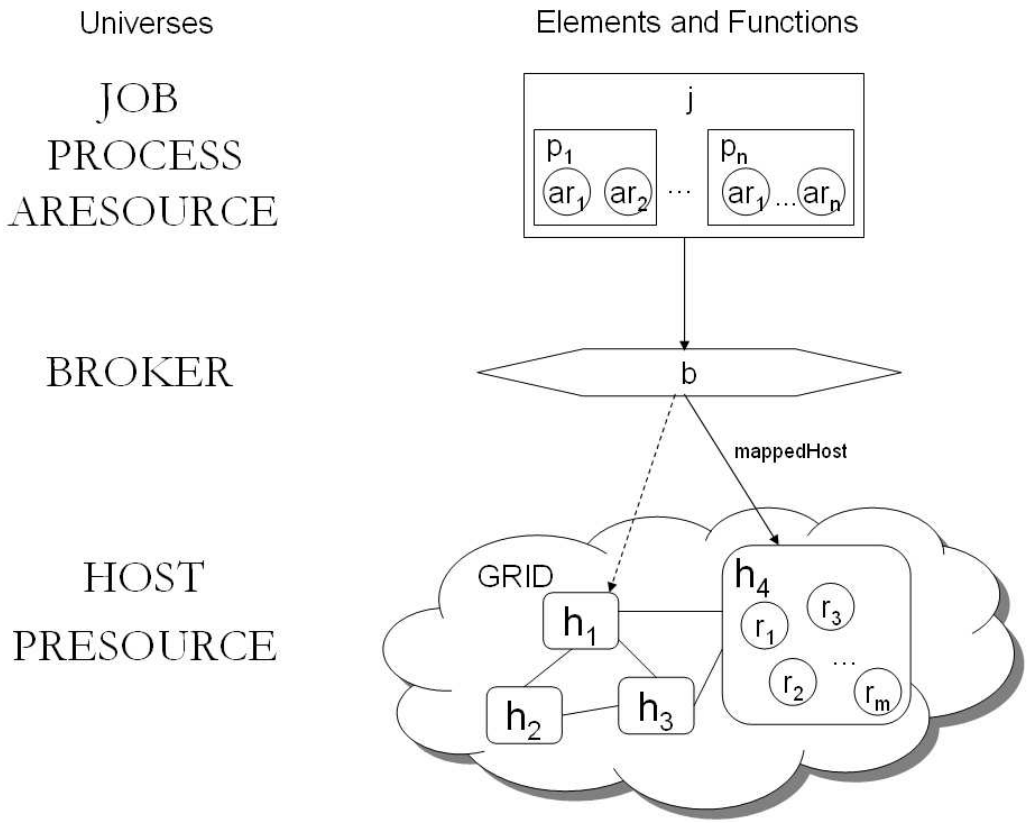

Figure 3.5: Grid brokering in the ASM model.

$\Pi_{\text{host\_mapping}}$

if ($\exists j \in JOB, \exists ar_1, \ldots, ar_m \in ARESOURCE, \exists pr_1, \ldots, pr_m \in PRESOURCE$):
  mappedHost(j) = $undef$ & request(j, ar_i) = $true, 1 \leq i \leq m$
    then choose h **in** $HOST$
      satisfying compatible(attr(ar_i), attr(pr_k))
      where belongsTo(pr_k, h) = $true, 1 \leq i, k \leq m$
        mappedhost(j) := h
    endchoose
endif

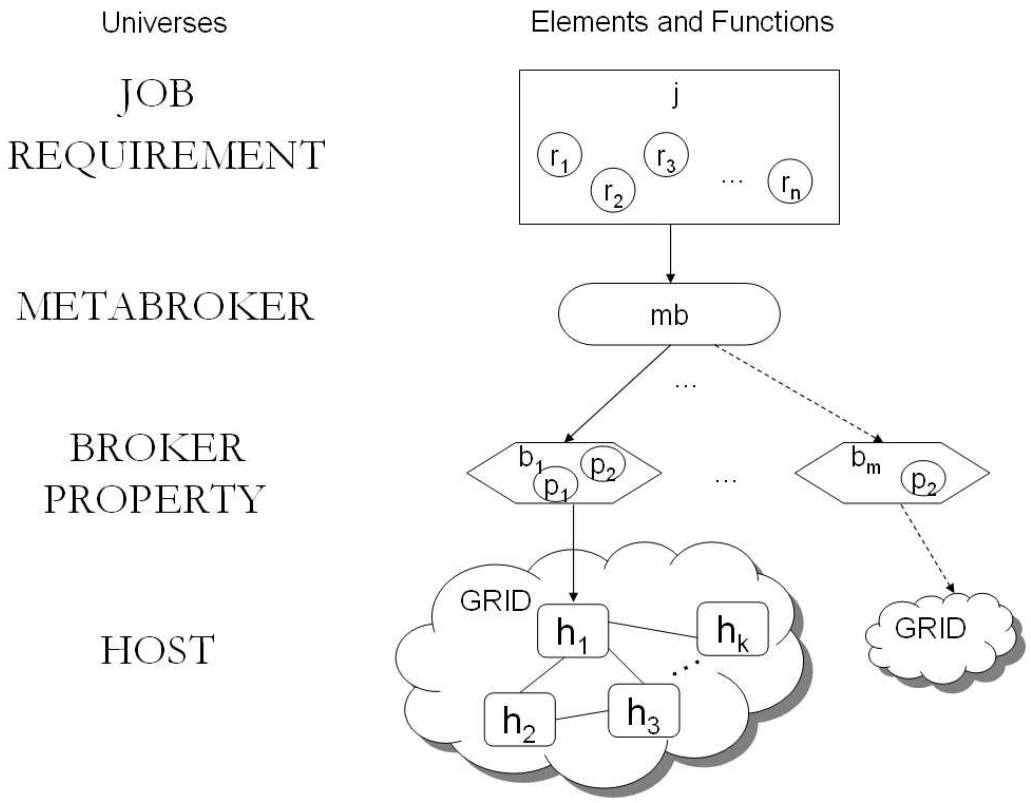

Figure 3.6: Meta-brokering in the ASM model.

3.3.2 Rule 5: Broker selection

At the highest level of Grid resource management a broker needs to be selected automatically for a user job. An important precondition of the selection process is that such a broker needs to be selected that manages hosts with resources that the user of the job can use. Furthermore the agent responsible for broker selection, the *meta-broker* (universe $METABROKER$) needs to ensure that the chosen broker has all the properties required by the user's job. Therefore users need to characterize their job requirements in a certain job description language, which should include both the required broker properties and abstract resources of the processes of the job. This additional Grid middleware component is highlighted in Figure 3.6. The following function is added to the model:

mappedBroker: $JOB \rightarrow BROKER$

I extend the initial state by:

$\forall j$: mappedBroker(j) = $undef$

The formal definition of the meta-broker is as follows:

let b = mappedBroker(j)
if ($\exists r \in REQUIREMENT, \exists pr \in PRESOURCE, \exists h \in HOST$):
  request(j, r) = $true$ & $b \neq undef$ & canUse(user(j), pr) = $true$,
  belongsTo(pr, h) = $true$, manages(b, h) = $true$
    then submitted(j, b) := $true$
endif

$\Pi_{\text{broker\_mapping}}$

if ($\exists r_1, \ldots, r_m \in REQUIREMENT, \exists p_1, \ldots, p_m \in PROPERTY, \exists j \in JOB$,
$\exists b \in BROKER$):
  mappedBroker(j) = $undef$ & $\forall i$: request(j, r_i) = $true$
  & $\forall i$: have(b, p_i) = $true$, $1 \leq i \leq m$
    then choose b **in** $BROKER$
      satisfying compatible(attr(r_i), attr(p_i)), $1 \leq i \leq m$
endif

Finally I have to mention that jobs can be interconnected in order to form a complex Grid application called a workflow. The execution of a workflow requires a coordinating tool called workflow enactor that schedules the interdependent jobs for executions. I refrain from formalizing workflow management and incorporate it into this model, since the central entities of this model are jobs, and therefore I assume that Grid applications are submitted into the system in the form of jobs.

A *state transition* diagram that shows the connections and graphically represents state evolving defined by Rule 2 and 3 together with the additional states of Rule 4 and 5 can be seen in Figure 3.7. The prerequisites of state changes are denoted by initials of the corresponding elements of the model (eg. $j \rightarrow b$ means the job is sent to a broker).

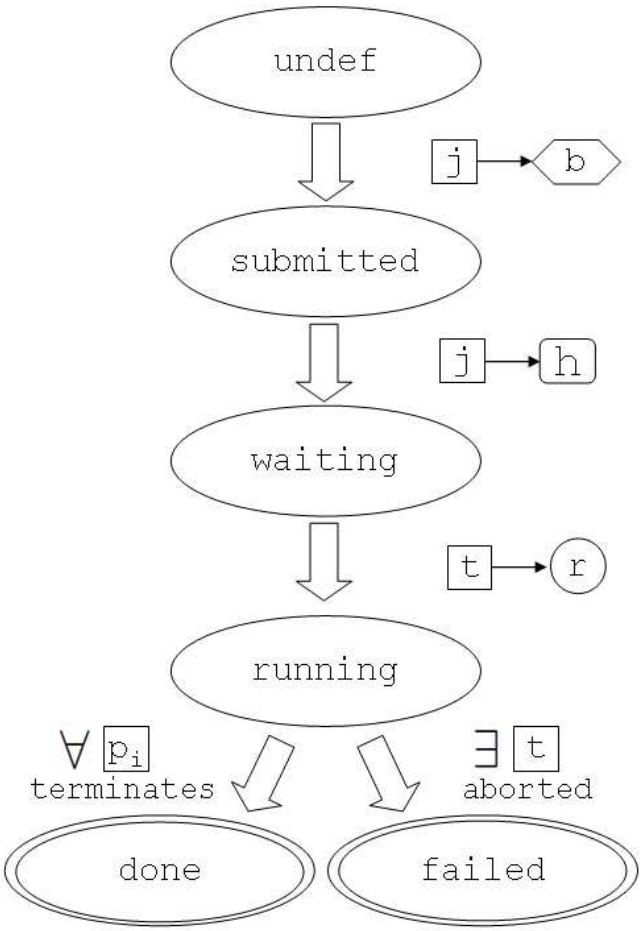

Figure 3.7: State transitions in the ASM model.

As a summary, I have shown that *Grid brokering* takes place at *three* levels, and the following operations need to be performed: broker mapping, host mapping and resource mapping. Later on I will show how practical examples of these components can be described by this formal ASM model with the help of ASM refinement. I will use this model to show how certain functions – kept abstract in Rule 4 and 5 presented earlier – are transformed to reveal implementation details.

3.3.3 Refining the ASM model to formalize the matchmaking of GTbroker

In the following I present the refinement of the host mapping (Rule 4) of the ASM model for Grid brokering introduced in Section 3.3.1.

$\Pi'_{\text{host\_mapping}}$

if ($\exists j \in JOB, \exists ar_1, \ldots, ar_n \in ARESOURCE, \exists policy \in REQUIREMENT,$
$\exists pr_1, \ldots, pr_m \in PRESOURCE, \exists h_1, \ldots, h_t \in HOST, \exists r_1, \ldots, r_t \in REAL$):
  mappedhost(j) = $undef$ & request(j, $policy$) = $true$,
  request(j, ar_i) = $true, 1 \leq i \leq n \leq m$
    then do forall k ($1 \leq k \leq t$)
      r_k := countRank($policy$, h_k)
      if ($\neg\exists l, i$): attr(ar_i) $\leq$ attr(pr_i)
      & belongsTo(pr_l, h_k) = $true, 1 \leq i \leq n, 1 \leq l \leq m$
        then $r_k := 0$
    enddo
    choose r_{max} **in** ($r_1, \ldots, r_t$)
      satisfying $r_{max} \geq r_k, 1 \leq k, max \leq t$
        mappedhost(j) := h_{max}
    endchoose
endif

This *refinement* also reveals the meaning of the *compatible* function. In case of GTbroker (discussed in Section 2.5.1), the attributes of resource requirements denote the amount of resource capacity (e.g. memory size or processor speed) needed by the processes of the job for execution. This means, if the available physical resource has equal or greater capacity than requested, the process can run. The host selection method can be influenced by users using the special *policy* requirement. The value of its attribute tells the additional countRank: $REQUIREMENT \times HOST \rightarrow REAL$ function how to compute the rank for the available hosts (e.g. higher priority can be given to hosts with faster processors). Finally, the host with the highest rank is selected for execution.

3.4 Interoperability levels for Grid brokering

In Section 2.4 we have seen that there are different interpretation models for interoperability in general in the literature, but these models cannot be applied to my

particularly focused area of Grid resource managers. To denote and differentiate the quality and degree of interoperability different solutions provide, I rely on the application requirements users pose on the execution environment. To satisfy these requirements, brokers need to find those resources that are able to execute the user jobs and need to provide special services (or capabilities). Hosts of different Grids may have different resources specialized for specific user application needs, and different brokers are specialized in different job execution policies and services (see the taxonomy and survey in Section 2.2 and 2.3). In order to categorize different Grid Interoperability solutions, I define the following *interoperability levels*:

No interoperability When a broker can access resources and services of only one particular Grid.

Low-level interoperability When a broker (or more inter-connected or centrally managed brokers) can reach resources of more, different Grids.

High-level interoperability When more inter-connected or centrally managed heterogeneous brokers can reach resources and different services of more, different Grids.

Using the previously defined ASM model for Grid brokering, I give a *formal* definition for these interoperability levels. In the model I have defined so far, I used the assumption that a broker manages hosts of only one Grid. In this case the type or identifier of a Grid could be expressed as a property of a broker. I also used the assumption that users can use a broker if they can access hosts managed by that broker (thus belonging to the same Grid). In order to express interoperability levels in my model, I extend it with a new set containing different Grids (universe $GRID$) that serve as host providers, therefore the following function is added to the model:

provides: $GRID \times HOST \rightarrow \{true, false\}$

I extend the initial state by:

$\forall h_i \in HOST, 1 \leq i \leq k : \exists g \in GRID : \text{provides}(g, h_i) = true$

As a simple definition I can state that Grid Interoperability means providing access to hosts of different Grids. Since most brokers are coupled to one Grid, therefore

it is able to provide access only to its hosts. This statement leads us to the 0-level *no interoperability*, which is formalized as follows. For a given $b \in BROKER$:

$\forall h_i \in HOST, 1 \leq i \leq k : \exists! g \in GRID :$
$\text{manages}(b, h_i) = true \ \& \ \text{provides}(g, h_i) = true$

The *low-level interoperability* means that a broker is able to provide access to hosts of more, different Grids:

$\forall g_i \in GRID, 1 < i \leq k : \exists h \in HOST :$
$\text{provides}(g_i, h) = true \ \& \ \text{manages}(b, h) = true$

In this case, if $k = 1$ for a broker b it means it has no interoperability, and if $k > 1$ it has a 2-level interoperability. Furthermore the higher the value of k is, the more interoperable the broker is.

In order to denote *higher degree* of interoperability, a more refined definition of Grid Interoperability is needed. Besides providing access to more Grids and therefore to a higher number of hosts, the other goal of interoperability is to provide support for more user requirements by utilizing (brokering) services of different Grids. This leads us to the definition of the *high-level interoperability*, in which we take into account the number of user requirements that a broker b and its managed hosts and resources may satisfy:

$\forall r_i \in REQUIREMENT, 1 \leq i \leq n : \ \exists g_l \in GRID, 1 \leq l \leq q,$
$\quad \exists h \in HOST, \exists pr \in PRESOURCE :$
$\text{manages}(b, h) = true \ \& \ \text{provides}(g_l, h) = true \ \&$
$\text{belongsTo}(pr, h) = true \ \& \ \text{compatible}(\text{attr}(r_i), \text{attr}(pr)) = true$

This expression tells us that a broker b is able to satisfy k user requirements by submitting the user job to a physical resource of a Grid that supports these requirements. Besides abstract resource requirements, user jobs usually need to use special broker services called broker properties (examples can be found in the broker taxonomy in 2.2). In order to determine the number of properties (or capabilities) a broker has we need the following expression:

$\forall r_j \in REQUIREMENT, 1 \leq j \leq m : \exists p \in PROPERTY :$
have(b, p) = $true$ & compatible(attr(r_j), attr(p)) = $true$

If all r_i and r_j are pairwise disjunct elements of universe $REQUIREMENT$, $k+m$ gives the total number of disjunct user requirements a broker b can satisfy. For a given meta-broker $mb \in METABROKER$ this number is the sum of the satisfiable disjunct requirements of all brokers it utilizes. As a result those meta-brokering solutions fall into this category that support different broker properties/services. The higher the degree of interoperability of a solution the higher the number of disjunct user requirements $(k+m)$ it is able to satisfy.

A *graphical* representation of the introduced interoperability levels can be seen in Figure 3.8. G_i represent different Grids, and b_i denote brokers having different capabilities.

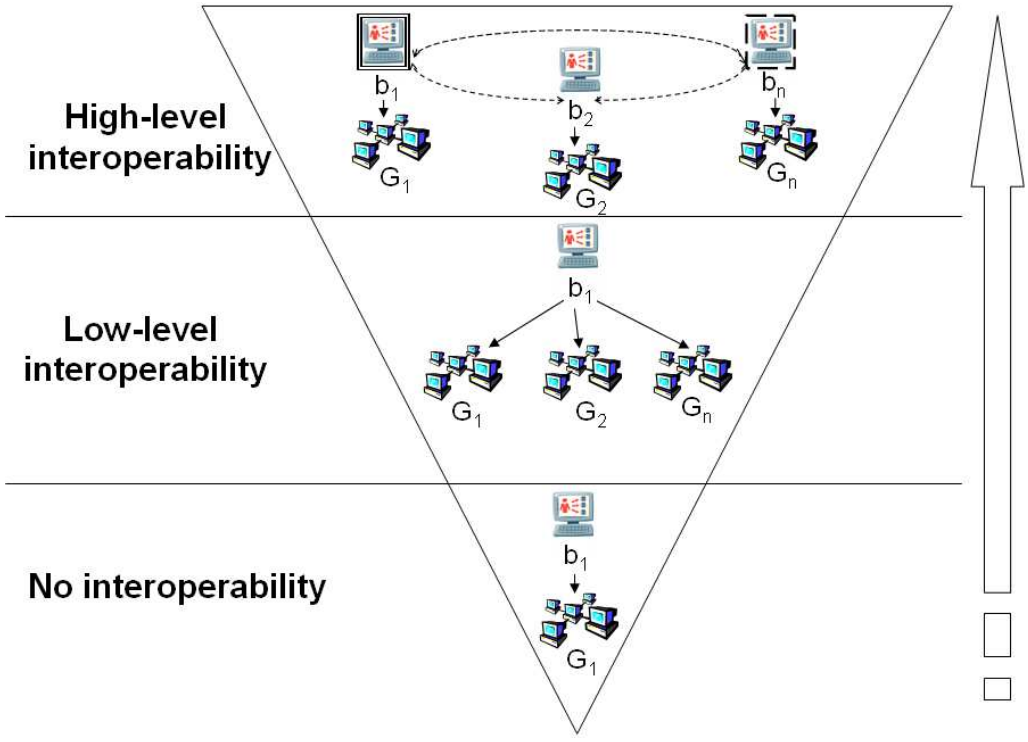

Figure 3.8: Interoperability levels.

3.5 Summary

In this chapter I investigated Grid resource manager components both informally and formally by developing an anatomy and an ASM model for Grid brokering. I highlighted how interoperation appears in this model and defined interoperability levels that will be used later to classify related solutions. The results of this chapter belong to thesis I, and were published in papers [P16], [P18] and [P19].

Chapter 4

High-level brokering solution for establishing Grid Interoperability

4.1 A general architecture for meta-brokering

In Section 2.4 we have seen that the Grid Interoperability problem can be tackled in short term by developing gateways that serve as entry services of Grids. Such a service should be built by using the latest standards and new technologies and trends. Regarding new technologies, the Next Generation Grids (NGG) Expert Group has identified a convergence between Grid and web services [61]. IT companies are developing and adapting their services to utility services, in which agent technologies, semantics, heuristics and self-awareness play a more important role taking into account the latest end-user requirements. They call these utility services Service Oriented Knowledge Utilities (SOKUs), which will become the building blocks of future Grids. This convergence of Grid services brings other technologies closer, and Grid development takes over new ideas and solutions from related research fields. Since this evolution takes much time to transform the whole system and there is a high demand for establishing Grid Interoperability, I have been looking for a solution that requires minimal or no modifications at all to the middleware, and still incorporates new technologies having the ability to become a SOKU in future Grids. In this section I introduce a novel scheduling philosophy called meta-brokering that creates a meta-level above current resource management solutions by using these technologies and open standards. Following this way, I have developed a method to make data about resource managers available for cooperated, automatic processing in the form

of meta-data. I provide language schemas to store and share this meta-data, and to be processed by various scheduling policies. First I examine the requirements of this meta-brokering approach, then present a general solution that can be realized in different Grid environments.

Interoperability problems appeared in several parts of Grid middleware, including resource management. Since most of the middleware components rely on or are in connection with resource managers, it is crucial to establish interoperation among different Grid resource managers. The lack of standards is a problem that cannot be solved in shorter period of time, but there is a definite need for standardization to tackle the interoperability problem. The proper solution for interoperability is to design a system, in which all components know how to communicate with each other. Protocols need to be established and utilized by all components. However, when protocols are diverse by themselves, the classical approach to this problem is to create a superior instance, a kind of mediator in a sense, which has the task to provide an interface between the individual components. This approach is taken in my case to create a new layer in Grid resource management called *meta-brokering* (denoted by MB in Section 3.1). The role of this layer is to utilize the existing, widely used and reliable resource brokers and to manage them transparently. Since most of the users have certificates to access more Grids (or Virtual Organizations (VOs)), they are facing a new problem: which Grid/VO, which broker should I choose for my specific application? Just like users needed resource brokers to choose proper resources within a Grid, now they need a meta-brokering service to decide, which broker (or Grid/VO) is the best for them and also to hide the differences of utilizing them.

Figure 3.1 in Section 3.1 shows the relevant actors in Grid resource management; we need to examine this anatomy of Grid resource management in order to find the right solution. Below the RMS level standardization could be one solution, since this lower layer is part of the core of every middleware system, therefore all of them would need to be redesigned to support a common standard. A good candidate for this solution is virtualization, which has already attracted several researchers and a new research area was born called Cloud Computing [14]. Above the RMS level each component could actually help enhancing interoperability. Starting from the user interface, the diversity in job description languages represents the first problem: though the Job Submission Description Language (JSDL) [113] is a standardized language, only few systems have adopted it, yet. Starting from the highest level, Grid portals are the first candidates for solving the Grid Interoperability problem. Some of them

have already achieved some level of interoperability [45], but for the final solution, the support for all the job description languages, interfacing all resources, RAMs, RMSs, WFMSs and related middleware services should be done, which cannot be performed with the current approaches. They need the help of some new component that takes up some of these duties. One solution could be a general WFMS that supports all the workflow description formats [106], interfaces the RMSs or RAMs and can be integrated to every portal. This seems to be an achievable solution, since only some services need to be supported, but still many descriptions have to be learned, and adapting such a WFMS would also need high efforts from the portal developers. This argument has led to the design of a *new meta-brokering service*. It is high-level enough to hide and manage the RMSs, and low-level enough to be integrated into portals and WFMSs.

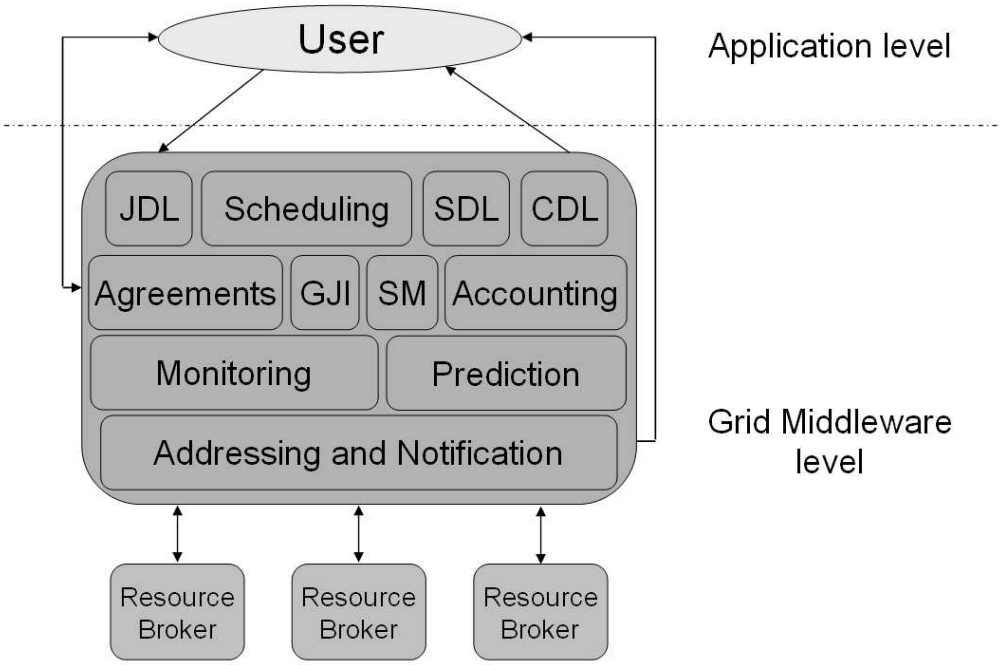

Figure 4.1: General meta-brokering architecture

Now that I have pointed out the place where a new abstraction layer responsible for establishing interoperability could be created, I continue with the requirements of this solution that fills the gap among the different components in resource management. Figure 4.1 is intended to show all the components and tools needed by a general Meta-Broker. In the following I describe these components by introducing the main

requirements of this higher level brokering service:

JDL – Job Description Language: Since the goal of a meta-brokering service is to offer a uniform way to access various Grids, a unified description format is needed to specify all the user requirements. JSDL [113] proposed by OGF is a good candidate.

CDL – Capability Description Language: Each broker has a different set of functionalities, they can be specialized in different application types. In order to store and track these properties, it is required to use a CDL. It should be general enough to include all the service capabilities (interfaces, job submission, monitoring and agreements).

SDL – Scheduling Description Language: Besides CDL and JDL the scheduling requirements and policies also need to be stored separately. The users can express their needs by extending the JDL with SDL, and the scheduling policies and methods of the brokers can be stored in this format.

Scheduling: This component performs the scheduling (matchmaking) of incoming user requests. A proper Grid broker (which implies a domain, VO or execution environment) needs to be selected for a user job taking into account the available scheduling policies.

GJI – Global Job Identifiers: It is important to have unique mapping of user jobs to different Grids. An implementation can be a single job ID provider for the meta-brokering system or simply using each broker system as a prefix for the assigned Grid job ID.

SM – Security Management: The role of this component is to provide secure access to the interconnected domains. For example, different user certificates, proxies may be accepted in different VOs and Grids. In order to provide a transparent way for users, these various proxies also need to be handled by meta-brokers.

Accounting Mechanism: The GJI and SM can be a part of a global accounting component. The role of this mechanism is to manage user access by pre-defined policies. Though Grid economy is still in a pre-mature state, in the future the meta-brokering service should also serve business Grids or clouds.

Agreements Mechanism: This component is in connection with the Accounting mechanism. Service Level Agreements (SLA) are planned to be used in future Grids, though investigating SLA usage in Grids has already been started [65]. The role of this part is to negotiate user requirements, which can also affect scheduling policies. When agreements will be generally accepted and used, this mechanism should be

extended to do negotiations with higher and lower levels.

Monitoring Mechanism: Reliable operation requires global monitoring, in terms of the inter-connected brokers, reachable domain, Grid resources, load and local component functionalities. Self-awareness and fault tolerance need to be provided by the system itself, which needs extensive monitoring.

Prediction Mechanism: This component is in connection with the Monitoring and Scheduling mechanisms. It is necessary to perform calculations of broker availability, service utilization and user request load to cope with the highly dynamic nature of Grids.

Addressing and Notification Mechanism: This component is responsible for accessing the inter-connected resource brokers, and managing communication including local events and external job state notifications.

The goal of the presented meta-brokering approach is to establish a connection between individual Grids (domains or Virtual Organizations) by managing their brokers. The general meta-brokering architecture (Figure 4.1) is middleware-independent, therefore implementations of this framework can solve the Grid Interoperability problem and it can be easily interfaced by application level tools, such as portals or workflow managers. The next section discusses how such an implementation can be realized in a service called Grid Meta-Brokering Service (GMBS).

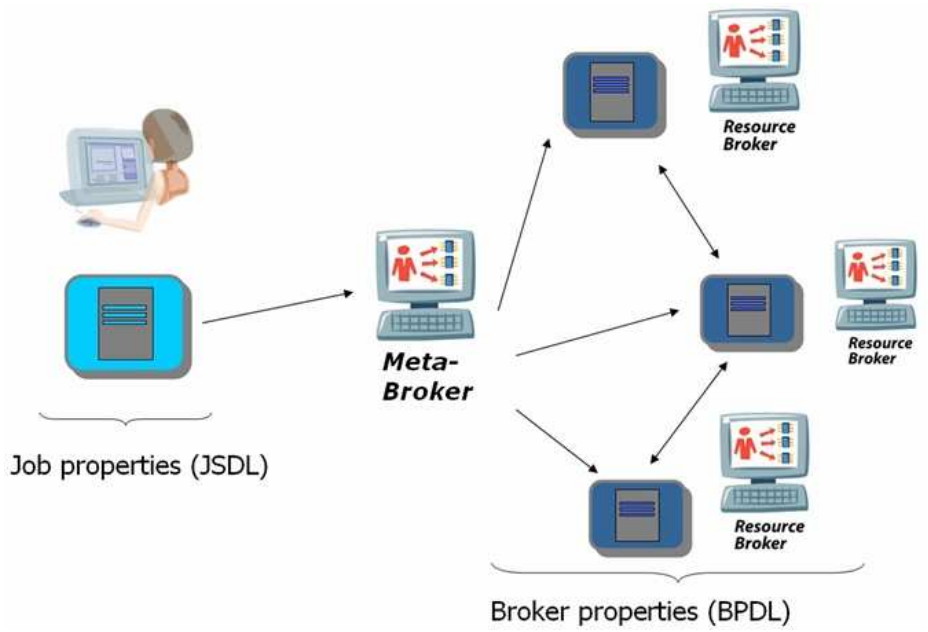

Figure 4.2: Description languages for meta-brokering.

4.2 Grid Meta-Brokering Service for high-level resource management

Figure 4.2 depicts the necessary description documents need to be used in order to facilitate meta-brokering. Heterogeneity appeared not only in the fabric layer of Grids, but also in the middleware. Even components and services of the same middleware may support different ways for accessing them. After a point this variety slows down Grid development, and makes the system unmanageable.

Table 4.1: A subset of special job description language attributes.

RSL (GTbroker)	*xRSL (NorduGrid)*	*JDL (EGEE)*	*JSDL*
(*sched=random*)	(*sched=random*)	FuzzyRank=true;	extension
(*sched=CPU/Memory/Disk*)	(*sched=CPU/Memory/Disk*)	rank=other.GlueHostProcessorClockSpeed/GlueHostMainMemoryRAMSize/GlueSAStateAvailableSpace;	extension
(*minMemory=int*), (*mindisk=int*)	(memory=int), (disk=int)	Requirements: (GlueHostMainMemoryRAMSize<int); anyMatch(other.storage.CloseSEs, target.GlueSAStateAvailableSpace>int);	⟨resources⟩ ⟨jsdl:IndividualDiskSpace⟩ ⟨jsdl:IndividualPhysicalMemory⟩ ... ⟨/resources ⟩
(*skiptime=int*)	(*skiptime=int*)	/*skiptime=int*/	extension
rescheduling by default	(rerun=max.5)	RetryCount=max.10;	extension

Languages are one of the most important factors of communication. Different resource management systems use different resource specification descriptions like RSL, JDL, etc. These documents need to be written by the users to specify all kinds of job-related requirements and data. The OGF [124] has already started to take several steps towards interoperability by defining standards, and developed a resource specification language standard called JSDL [113] – this should be used to overcome the above mentioned difficulties (with possible extensions). As the JSDL is also general enough to describe jobs of different Grids, I have chosen this to be the job

description format of my meta-brokering solution. Nevertheless there are special job attributes denoting special job handling, various scheduling and data management features that cannot be expressed in JSDL. Since one of the goals of meta-brokering is to support all these brokering capabilities, I gathered these special attributes of the different job description documents, and specified a JSDL extension – depicted in Figure 4.3 – that I also proposed to the Grid Scheduling Architecture research group of OGF [124] for standardization as an SDL. During the translation of these special requirements to languages that are not able to express them, the missing attributes are included as comments in order to keep the translations consistent. Some examples on the mapping among these attributes are shown in Table 4.1.

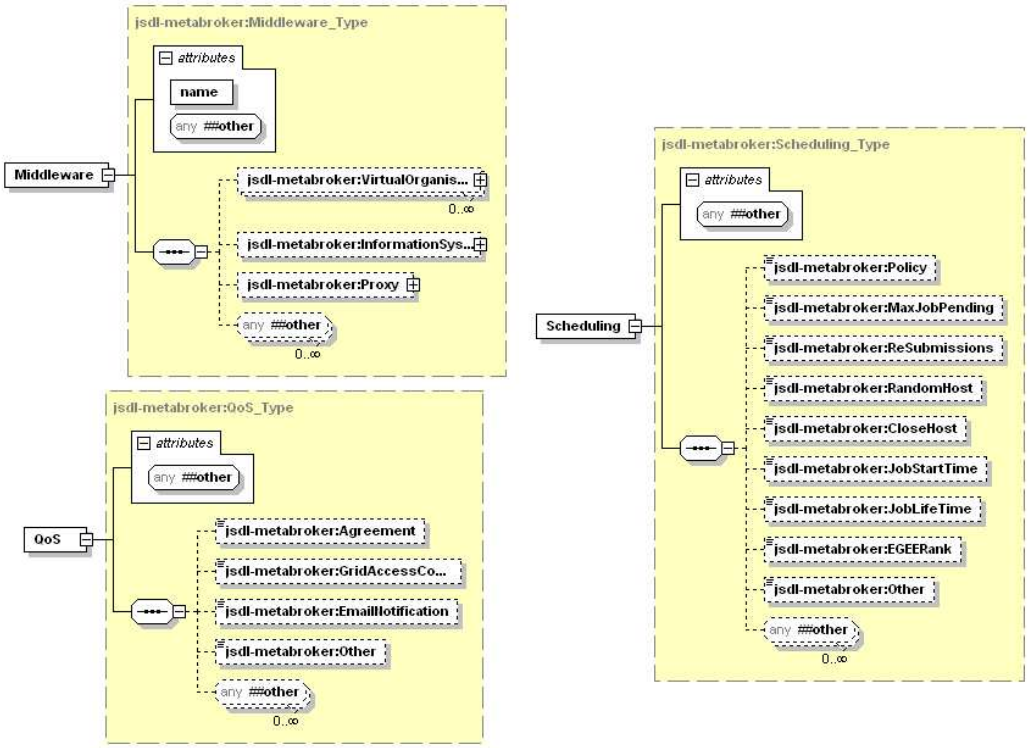

Figure 4.3: Main fields of the preliminary JSDL extension.

Besides describing user jobs, we also need to describe resource brokers in order to differentiate and manage them. These brokers have various features for supporting different user needs. These needs should be expressed in the user's JSDL, and identified by the Meta-Broker for each corresponding broker. Therefore I proposed an extendible *Broker Property Description Language* (BPDL) – similar to the purpose of JSDL –, to express metadata about brokers. This description language is discussed

in the following subsection.

4.2.1 Data Model for describing broker capabilities

For describing Grid Resource Broker capabilities, I introduce an extensible metadata model. This model can be taken as an extension of the general scheduling model presented in [67]. Beside their resource and job model, there is a need for a model describing broker characteristics in order to compare, interoperate and manage different resource brokers, schedulers. I use the same notations for building up the model.

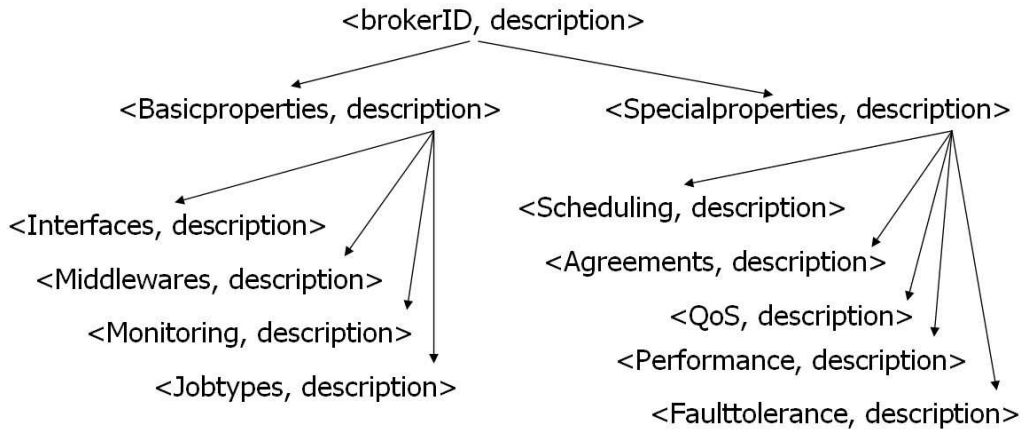

Figure 4.4: Structure of the data model for resource broker capabilities.

The metadata to be stored regarding resource brokers are expressed through <attribute,value> pairs – I denote with $\mathcal{P}$ the set of all possible such pairs. A broker denoted by $\mathcal{B} \subseteq \mathcal{P}$ is modeled as a pair:

<brokerID,description>,

where brokerID is a unique identifier, and description $\subseteq \mathcal{P}$ is a set of attribute/value pairs, which contains metadata of basic and special properties. Figure 4.4 shows the tree of pairs in $\mathcal{P}$, which defines the whole model.

In order to present a usage scenario for matchmaking, I define a function over this model with the following structure:

- μ: $\mathcal{T} \times \mathcal{B}^{\,i} \rightarrow \mathcal{B}$, where $\mathcal{T}$ is a set of tasks [67] (here: jobs) and $\mathcal{B}$ is a set of brokers.

For $t \in \mathcal{T}$, b_0, ... ,$b_n \in \mathcal{B}$, n≥0:

- $\mu(t, (b_1, b_2, b_3)) = b_2$ means that for a job denoted by t matched with brokers denoted by b_1, b_2 and b_3 the matchmaking function returns b_2, which is the fittest broker for the job. That means the returned broker can most efficiently execute the job. (Note that b_0 can be a special element, which is an empty description. This is the return value, when no broker fits the job requirements.)

In the scenario shown in the following section a JSDL of the job is denoted by t, and a BPDL of a broker by b_i.

4.2.2 The implemented data model: Broker Property Description Language

Based on the data model introduced in the previous section I have created an XML-based language called BPDL (Broker Property Description Language). The common subset of the individual broker properties are the basic properties: the supported middleware, job types, certificates, interfaces and monitoring issues. There are also special ones, such as remote file handling, fault tolerant features, agreement support, QoS support, performance metrics and various scheduling policies. The union of these properties forms a complete broker description document that can be filled out and regularly updated for each utilized resource broker – the graphical representation of this document can be seen in Figure 4.5. The special any##other type describes a mechanism that can be used to extend the schema with custom elements and attributes. The fields of the BPDL are closely related to the categories of the broker taxonomy presented in Section 2.2 of Chapter 2. This ensures that all the widespread brokers can be described with this language, therefore they can be managed by solutions using BPDL. Notice that this language can also be used for peer-to-peer communication and identification in a decentralized architecture. In particular, the agreements are another mechanism typically used in this kind of architectures to broaden a domain or as a communication mechanism during the negotiation process. In BPDL, the common subset of the individual broker properties is stored here:

- BrokerID: It contains a unique identifier of a resource broker.
- Interface: This field provides meta-data about the accessibility and notification methods of the broker.

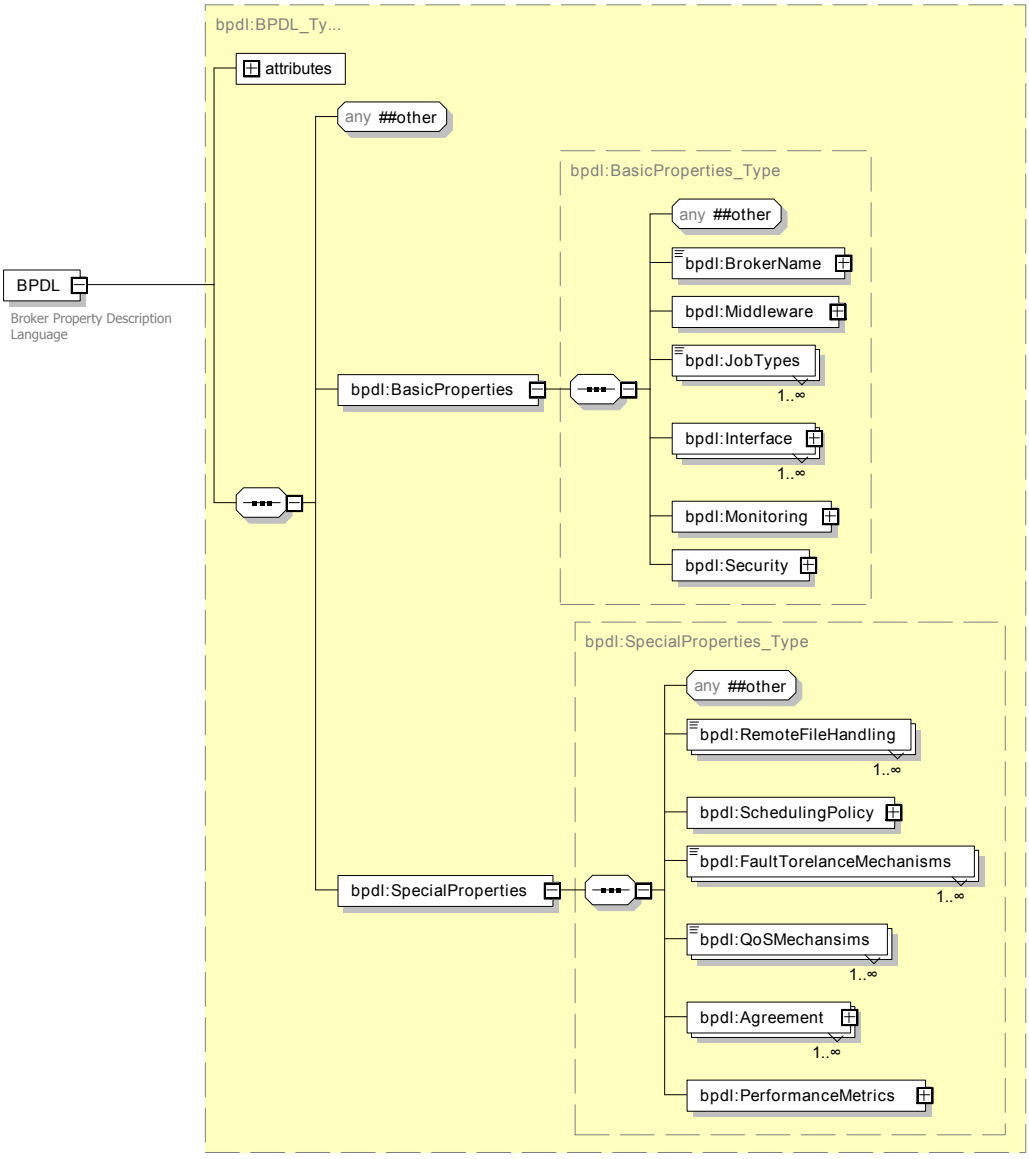

Figure 4.5: The schema of the Broker Property Description Language.

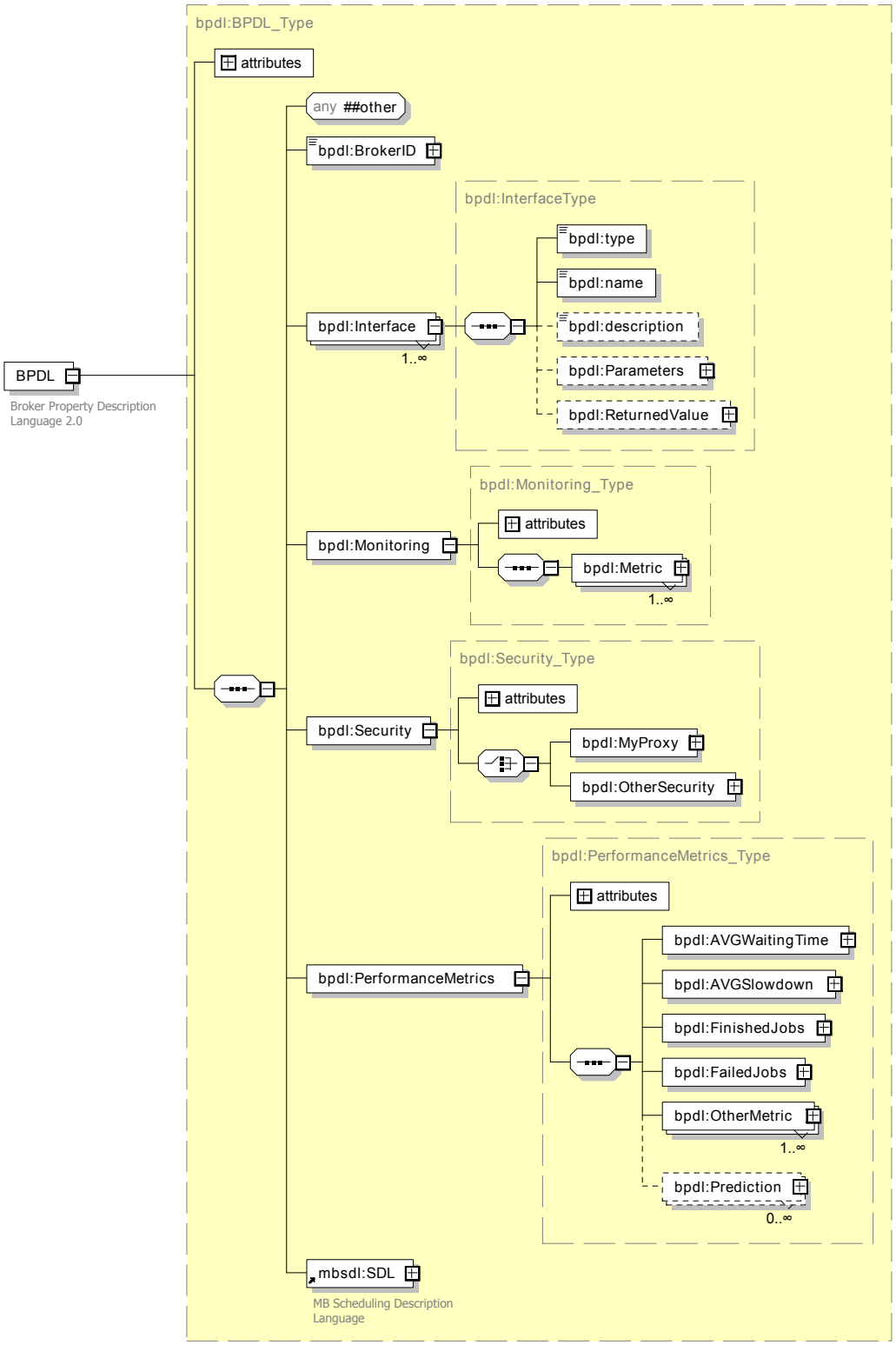

Figure 4.6: The schema of the Broker Property Description Language 2.0.

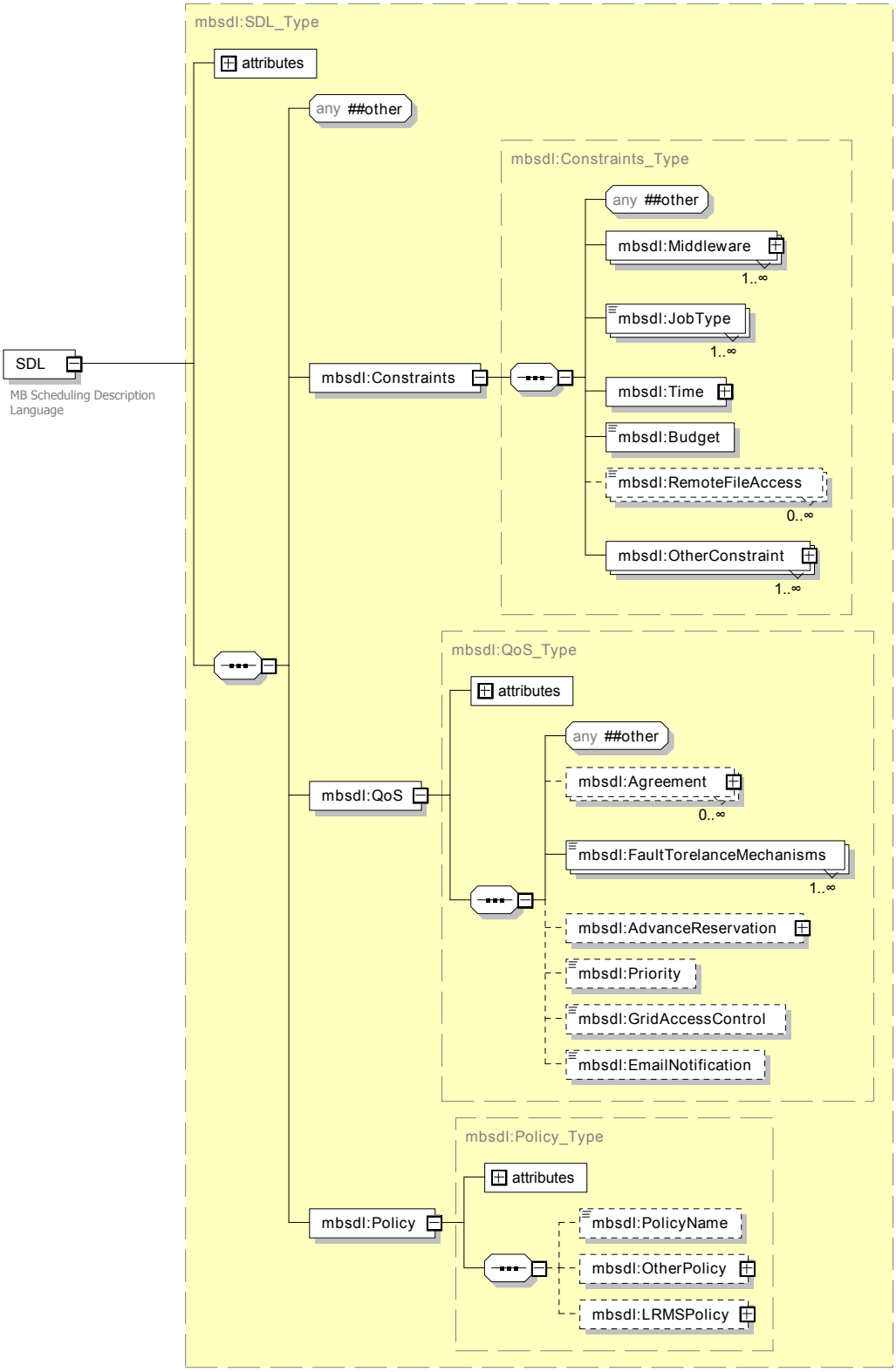

Figure 4.7: The schema of the Meta-Broker Scheduling Description Language.

- Monitoring: This field used for specifying self, job or resource monitoring mechanisms of the broker.

- Security: It provides data about job and user authentication methods, such as MyProxy server connections.

The most important attributes storing static information are the followings:

- Middleware: It shows, in which kind of middleware, Grid or VO the broker can operate, which Information Services it uses.

- JobType: It specifies the type of jobs that the broker can handle.

- RemoteFileAccess: This field shows the protocols used for transferring files.

The dynamic information is updated by the GMBS during utilization. This data is intended to provide up-to-date performance and availability information for scheduling. The following field is used for this purpose:

- PerformanceMetrics: It stores historical data on previous job submissions, which can be used to determine reliability of broker properties. The Prediction attribute can be used to store predicted data about broker availability and reliability.

After I realized that the OGF standardization process may take several years to come up with a commonly accepted SDL, I decided to revise and modify BPDL by gathering the scheduling-related attributes to a revised JSDL extension called *MBSDL* (Meta-Broker Scheduling Description Language). The structure of the new BPDL – that I call BPDL 2.0 –, remains nearly the same, I have only clarified some attributes, added missing ones and separated the scheduling-related ones to MBSDL. Therefore the MBSDL language can be used to extend BPDL with scheduling-related attributes. The graphical representation of BPDL 2.0 can be seen in Figure 4.6, and the same of MBSDL in Figure 4.7. The full schema of these XML documents can be seen in the appendix in Chapter C. Since JSDL is also lacking these attributes, MBSDL can also be used by other brokers or resource manager tools as a JSDL extension. Its schema contains three fields:

- Constraints: In this field we can specify terms that are necessary to be fulfilled during scheduling. It includes middleware, remote file access and job type constraints, as well as processing time and budget cost requirements. Finally there is an opportunity to specify customized ones.
- QoS (Quality of Service) requirements: Here one can specify agreements, job priorities, advance reservations, email notification or access controls. Fault tolerant features can also be selected to affect the schedule.
- Policy: In this field we can choose from various scheduling policies, or we can define customized ones. E.g. the LRMSPolicy is used to describe local scheduler capabilities.

The union of these properties forms a complete broker description document that can be filled in and regularly updated for each utilized resource broker. These two kinds of data formats are used by GMBS (Figure 4.2): JSDL is used by the users to specify jobs and the BPDL by administrators to specify brokers – both parties can use MBSDL to extend their descriptions. To advance standardization processes in this topic I keep on negotiating with the GSA-RG (Grid Scheduling Architecture Research-Group) of OGF [124] in order to create a common Scheduling Description Language. Once it is done, I will use that instead of MBSDL.

4.2.3 Description of the components of GMBS

Figure 4.8 introduces the derived architecture (from the general architecture shown in Figure 4.1) of the meta-broker I propose to solve Grid Interoperability at the resource management layer of Grids. This figure can be compared to the general meta-brokering architecture, and it gives an insight which abstract components are realized in this solution. Figure 4.9 (a refined version of Figure 4.8) introduces the final architecture of the *Grid Meta-Broker Service* (GMBS) that enables the users to access resources of different Grids through their own brokers. In this way, this higher level tool matches resource brokers to user requests. The system is implemented as a web-service that is independent from middleware-specific components. The provided services can be reached through web interfaces defined by WSDL. A UML class diagram representing the main components of GMBS can be seen in Figure C.1 of the appendix in Chapter C. In the following I describe the role of its components and their interaction.

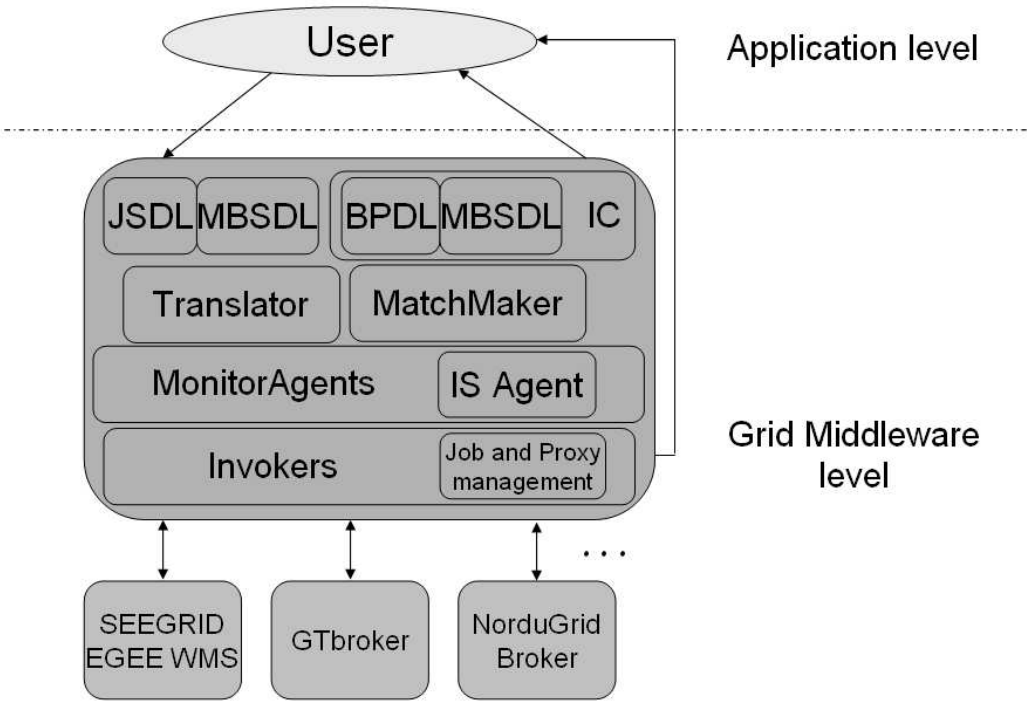

Figure 4.8: Realization of the general architecture.

When I first presented this novel Grid meta-brokering approach in 2006 [P3], there were not any other related solutions. Later this approach has matured and I created the first prototype and published more detailed papers about this meta-brokering solution. Meanwhile other research groups have also realized the need for the meta-brokering approach and started to develop their own solutions, which appeared in the literature, and contained citations to my publications. A summary and classification of these related works are given in Section 4.3.

The *Translator* component of GMBS is responsible for transforming the resource specification defined by the user (in JSDL and MBSDL) to the language of the appropriate resource broker that the meta-broker selects to use for a given job. From all the various job specification formats a subset of basic job attributes can be chosen, which can be denoted relatively in the same way in each document (these attributes also exist in JSDL). The translation of these parts is almost trivial. The rest of the job attributes describe special job handling, various scheduling features and remote storage access. Generally these cases can hardly be matched among the different systems, because only few of them support the same solutions (some examples are shown in Table 4.1), even the same functionality can be expressed in different ways

in different languages.

The *Information Collector* (IC) component stores the data of the reachable brokers and historical data of the previous submissions. This information shows whether the chosen broker is available, or how reliable its services are. During broker utilization the successful submissions and failures are tracked, and regarding these events a rank is modified for each special attribute in the BPDL of the appropriate broker (these attributes were listed above). In this way, the BPDL documents represent and store the dynamic states of the brokers. All data is stored in XML, and advanced XML-serialization techniques are used by the IC. The load of the resources utilized by the brokers is also taken into account to help the Matchmaker to select the proper environment for the actual job. When a large number of jobs with similar requirements are sent to the Meta-Broker, the so-called best effort matchmaking (choosing the less loaded one) may flood a broker and its utilized resources: that is the main reason, why load balancing is an important issue. In order to cope with this problem, there is an *IS Agent* service reporting to the Information Collector, which regularly checks the load of the Grids of each connected resource broker, and store this data. This tool is implemented as a separate web-service connected to the Information Systems of the Grids of the utilized brokers (the IS acronym denotes this role). On the contrary to the work performed by GIN-INFO [125], where contents of different information schemas are translated, the IS Agent creates and aggregated database of minimal attributes from these schemas, which is regularly updated. This approach require no modification or additional components for the utilized production Grids (unlike the GIN top-level-DBII approach [125]). With the additional information provided by this agent the matchmaking process can adapt to the load of the utilized Grids. Finally, the actual state (load, configurations) of the GMBS is also stored here, and it can also be queried by users. The continuous monitoring of Grid load and broker performances makes this Grid service self-adaptive.

Since the agreement handling and SLA usage are not mature enough in current Grids [65], the Accounting mechanism of the general architecture stated in Section 4.1 is not implemented in the GMBS, yet. Furthermore I think that SLA negotiation can be complex enough to be managed by a separate service that would cooperate with GMBS in agreement negotiation processes. On the other hand, the MBSDL language provides a similar solution: it is possible to denote in the *QoS requirements* and *Policy* fields some special requirements that are forwarded by the GMBS to the selected broker (that supports such requirements). Then it will be the task of the

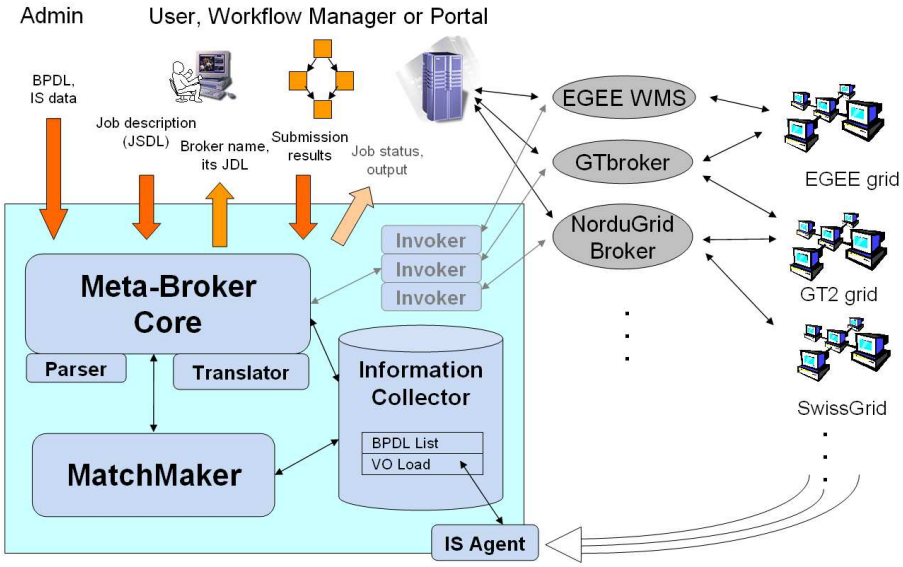

Figure 4.9: Grid Meta-Broker Service

invoked broker to ensure the QoS requirement during the actual resource selection. The Security and Accounting mechanisms are not implemented in the GMBS, either. The security solution applied in this service is the same as described in [45] for the multi-grid P-GRADE portal and hence I do not give more details here on the security issues. There are two scenarios how to use GMBS:

Scenario 1: Figure 4.10 depicts a sequence diagram for job submissions through the GMBS. When the users or portals prefer to invoke and track the brokers themselves, the Invoker component of the Service is not used. In this case only the JSDL document of a job needs to be provided for the GMBS. First the *Core* (MBService) calls the *Parser* to get the job details then turns to the *MatchMaker* component to match the required services to the properties of the brokers stored in the Information Collector. When the fittest broker is found, it contacts the Translator. After the Translator has converted the JSDL to the job description language of the matched broker, it responds with the name of the broker and its job description (or with a message that none of the registered brokers is able to fulfil the specified job requirements). In this case security issues are handled by the user or the portal (just like ordinary job submissions). Finally, the output of the submission (e.g. execution time, success/-failed status, etc.) needs to be provided to the GMBS (to let the Core modify the broker property ranks). Here I note that according to this scenario description, the GMBS only knows about jobs submitted to the Grids through its interface, therefore

it does not have a global view of all the reachable Grids. Nevertheless it is prepared for this uncertainty, and it relies more on the reported historical job submission results. In the simulation environment (to be discussed in 5.1.2) I use real workloads, which submissions are done directly to the brokers without the use of the GMBS. These jobs increase the load of the simulated Grids and appear in the information system of the simulation environment (just like in real Grids), which is tracked by the GMBS, and the load information is used during matchmaking. Scenario 1 is useful for systems that already have reliable connections to resource brokers and would like to use the meta-brokering service for broker selection and inter-Grid load balancing. Currently these issues are not taken into account in Grid portals. Even multi-grid access is rarely supported, where the users have to choose from a list of resource brokers. Furthermore this utilization can be achieved with minimal adaptation efforts and requires fewer data transfers.

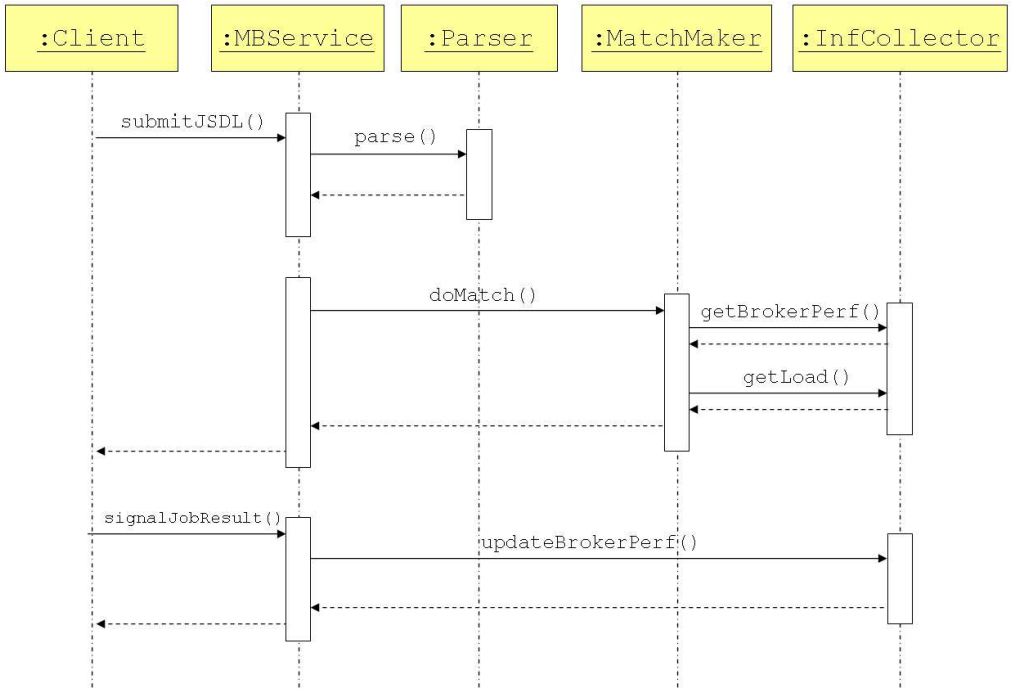

Figure 4.10: Sequence diagram of the GMBS utilization.

Scenario 2: When the actual job submission to the resource brokers is also done by the GMBS, the *Invoker* components are used to contact the brokers. The Invokers are broker-specific components: they communicate with the interconnected brokers, invoking them with job requests and collecting the results. In this case data handling

is also an important task of this component. In this case the user has to upload the job, Grid certificate proxies and input files along with the JSDL to the GMBS, and the Matchmaker component tries to find a proper broker for the request. If it could not find a broker that was able to fulfil the user requirements, the request is discarded, otherwise the JSDL is translated to the language of the selected broker. In the JSDL extension the middleware constraint fields can be used to specify certificate proxy names for Grids/VOs. This information is used by the Invokers to select the valid certificate proxy from the uploaded files for the actual job submission. Then the responsible Invoker takes care of transferring the necessary files to the selected Grid environment. After job submission, it stages back the output files and upgrades the historical data stored in the Information Collector with the log of the utilized broker. The Core component of the service is responsible for managing the communication (information and data exchange) among the other components. The communication to the outer world is also done by this part through its web-service interface. Generally the following operations can be done through this interface (see Table 4.2): adding a new broker with BPDL, querying the available brokers and the name of the tracked Grids/VOs (by IS Agents), adding new Information Systems to be tracked (by IS Agents), submitting jobs (with JSDL) and signaling submitted job results.

Table 4.2: Web Service interface methods of the GMBS.

WS operations	**Description**
addBroker	Adding a new broker to the system with its BPDL description
addVO	Adding a new VO or Grid to the system with its IS interface data
submitJSDL	Submitting a job with its JSDL description
signalJobResult	Notifying the MB of the result of the submission
getBrokerNames	Querying the utilized brokers
getBrokerPerf	Querying the performance rank of a utilized broker
getVONames	Querying the connected Grids or VOs (behind the brokers)
getVOLoad	Querying the actual load of a connected Grid or VO

Figure 4.11 shows the differences between these two scenarios. In the first case

(which corresponds to scenario 1 described above), a lightweight version of the GMBS is used, so the service can focus on scheduling and let the operator parties do the broker invocation. In the second case (scenario 2), the Invokers contact the brokers and take care of data movements.

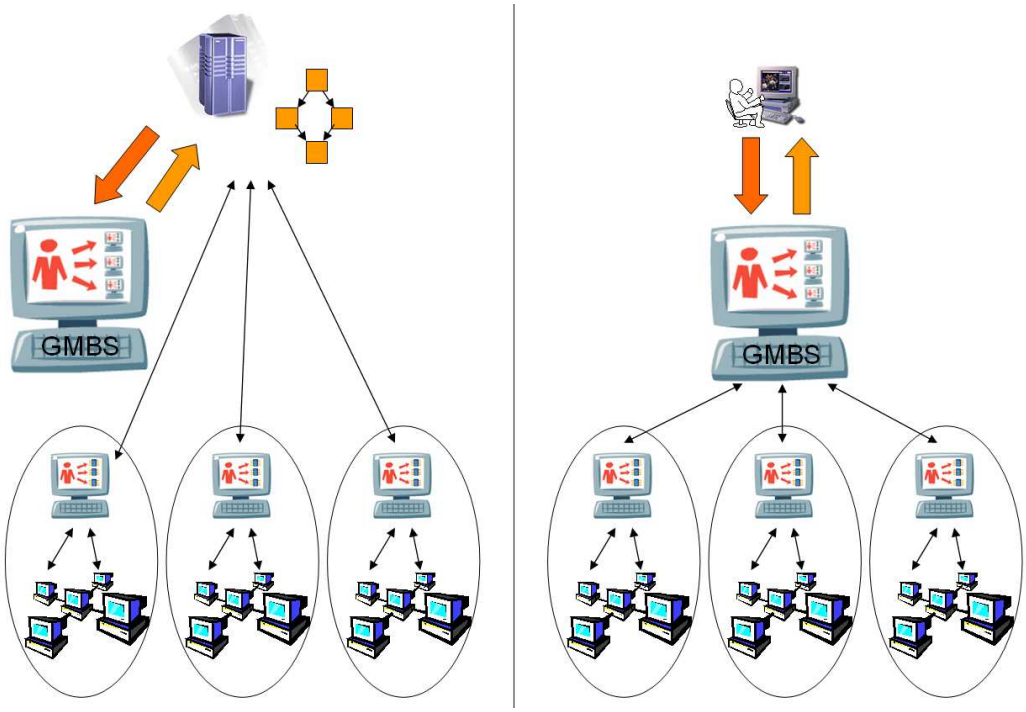

Figure 4.11: GMBS usage scenarios.

The previously introduced languages are used for matching the user requests to the description of the interconnected brokers: which is the role of the *MatchMaker* component. The scheduling strategy of GMBS is a non-queue-based one. It examines every user request in the same order of arrival in the system, which reflects the notion of fairness which is generally accepted by most users. Regarding broker selection, the used strategy is similar to the Condor matchmaking approach [82] that has been widely used in Grid resource management. The matchmaking process of GMBS also takes into account user interests and systems performance. The JSDL contains the user request (this supposed to be an exact specification of the user's job) including the special attributes defined in MBSDL, while the interconnected brokers are described by their BPDL documents. The matchmaking process consists of the following steps to find the fittest broker:

- The Matchmaker compares the JSDL of the actual job to the BPDL of the

registered resource brokers. First the job requirement attributes are matched against the broker properties stored in their BPDLs: this selection determines a group of brokers that are able to submit the job (denoted by GOODBROKERS in Listings 4.1). This phase consists of two steps: first the brokers are filtered by all the requirements stated in the JSDL. If no brokers could fulfil the request, another filtering process is started with minimal requirements (those ones are kept, which are real necessary for job execution). If the available brokers still could not accept the job, it is rejected.

- In the second phase the previous submissions of the brokers and the load of the underlying Grids are taken into account. The pseudo code of the matchmaking functions of this phase is shown in Listings 4.1. The MatchMaker component counts a rank for each of the remaining brokers (by getBrokerPerf). This rank is calculated from the job completion rate (counted with getFinishedJobs and getFailedJobs) that is updated in the PerformanceMetrics field of the BPDL for each broker. The list of the remaining brokers is compared according to these ranks. Finally the first broker having the highest performance value is selected for submission, unless the background Grid load (queried by getVOLoad) of the second one is at least 20 percent less (this load is regularly updated by the IS Agent). In this later case the second best performing broker is selected.

This is the default scheduling used by GMBS. Different threshold values for load balancing among the candidate brokers may be used for better scheduling performance. More sophisticated scheduling algorithms with various fuzzy functions can be found in joint publications [P13, 19, 20]. We can see from the description of the matchmaking process that the GMBS does not use predicted data, but relies on its measured historical performance results of the brokers to cope with uncertainty. In the next section we will see simulation measurements for the makespan of groups of jobs. One may wonder if this makespan could be estimated, or the makespan of some individual jobs could be predicted and this information could be used for a more efficient scheduling. Unfortunately in Grids it is really hard (almost impossible) to estimate the finish time of a job. For such an estimation, we would need the exact running time of the jobs to be submitted, the number of the jobs, the exact waiting time of the submitted job on the selected resource at the time of submission (this is the running time of the queued jobs) including the response time of the resource manager, and the exact time of the input/output file transfers. Most of the time the

Listing 4.1: Pseudo code of the matchmaking of GMBS

```
FUNCTION: selectBestBroker

IN: GOODBROKERS = b_1, ..., b_n, candidate brokers

OUT: BESTBROKER, the matched broker

BEGIN:

BESTBROKER = GOODBROKERS_1
SNDBROKER = GOODBROKERS_2
FOR i = 2 TO GOODBROKERS.size() {
        IF (getBrokerPerf(BESTBROKER) <
        getBrokerPerf(GOODBROKERS_i)) {
                SNDBROKER = BESTBROKER
                BESTBROKER = GOODBROKERS_i
        }
}
/* choose the second best performing if
at least 20 percent less loaded */
IF (getVOLoad(BESTBROKER) > getVOLoad(SNDBROKER+20) {
        BESTBROKER = SNDBROKER
}

RETURN: BESTBROKER

FUNCTION: getBrokerPerf

IN: BROKER, a broker

OUT: PERFVAL, the performance value of the broker

BEGIN:

PERFVAL = (getFinishedJobs(BROKER)+1) /
                (getFailedJobs(BROKER)*3)+1)

RETURN: PERFVAL
```

underlying Grid middleware does not provide enough information on its related components or services (local managers), and even the users cannot give exact estimations on the run time of their own jobs [56].

4.2.4 Refining the ASM model to formalize the matchmaking of GMBS

In the following I present the refinement of the broker mapping (Rule 5) of the ASM model for Grid brokering introduced in Section 3.2 of Chapter 3.

$\Pi'_{\text{broker\_mapping}}$

if ($\exists r_1, \ldots, r_m \in REQUIREMENT, \exists p_1, \ldots, p_n \in PROPERTY,$
$\exists j \in JOB, \exists b_1, \ldots, b_l \in BROKER, \exists v_1, \ldots, v_l \in REAL$):
  mappedbroker(j) = $undef$ & $\forall t$: request(j, r_t) = $true, 1 \leq t \leq m,$
  & $\forall i$: have(b_k, p_i) = $true, 1 \leq i \leq n, 1 \leq k \leq l$
    then do forall k $(1 \leq k \leq l)$
      v_k := getBrokerPerf(b_k)
      if ($\neg\exists t, i$): attr(r_t) = attr(p_i) & have(b_k, p_i) = $true, 1 \leq t \leq m, 1 \leq i \leq n$
        then $v_k := 0$
    enddo
    choose v_{max} **in** ($v_1, \ldots, v_l$)
      satisfying $v_{max} \geq v_k, 1 \leq k, max \leq l$
        mappedbroker(j) := b_{max}
    endchoose
endif

This refinement also details how the *compatible* function is implemented. In case of the Grid Meta-broker Service, the attributes of the broker properties are certain keywords. The users have to use the same keywords in their requirement specifications, therefore compatibility means exact string matching. The refined agent also uses an additional function getBrokerPerf: $BROKER \rightarrow REAL$, which returns a real number denoting the dynamic performance of the appropriate broker. The higher this value is the better the broker performs.

In the next section I summarize the related works, and the next chapter continues with the performance analysis of GMBS.

4.3 Related Grid Interoperability efforts in Grid resource management

Several research groups have noticed that current Grid resource management tools will not be able to fulfil the high demands of future generation Grid systems, and have started to look for solutions to enhance interoperability. In the previous chapters I have introduced and formalized Grid brokering and the problem of Grid Interoperability. Though I propose interoperability solution at the resource management layer of Grid systems, I mention that there are interoperation efforts at lower and higher layers, too. For example, the Generic Grid-Grid Bridge (3G Bridge) [42] is a generic implementation for a low-level gateway service that allows the execution of jobs within different Grid middleware using Grid plugins that implement interfaces job submission entry points of different Grids. Regarding upper layers, a solution for Grid Interoperability at workflow level is presented in [45], which provides a way to execute the components of a workflow simultaneously in several Grids.

I differentiate three directions to tackle the problem of Grid Interoperability in Grid resource management:

1. the first approach is to extend existing brokers with multiple Grid middleware support,

2. the second direction uses portals to interface different brokers,

3. the third category represents higher level brokering approaches including inter-broker communication and meta-brokering, which introduces another level above current brokering solutions.

4.3.1 Related Grid Interoperability efforts with multiple Grid middleware support

The idea of this approach is to extend existing resource brokers with multiple Grid middleware support. We have already seen in the Grid broker taxonomy in Section 2.2

that some brokers are designed or have been extended to support services of different Grids. The following ones fall into this category:

- The Gridbus Grid Service Broker [86] is designed for computational and data-Grid applications. Although it supports all Globus middleware, and it provides an interface to be implemented for other middleware support (experimentally done for UNICORE [143] and Nordugrid [122]), it is mainly used in Globus Grids. In their latest publication [13] they also identified the burden of middleware support extensions and proposed a peer-to-peer resource discovery service to create a Grid-Federation network.

- Gridway [38] has been developed in a Globus incubation project, therefore it supports all Globus versions. Lately it has been extended to interface more middleware [105] (namely gLite, NorduGrid and OSG [127]).

- The JSS [24] is a decentralized resource broker that is able to utilize both GT4 [99] and NorduGrid resources, and they plan to interface EGEE.

- GTbroker has been designed for Globus Grids, later extended to support EGEE Grids. This broker will be introduced and further discussed in Section 2.5.1.

As we can see, some groups have started to extend their solutions, but the adoption of all services of other middleware systems could not be fully done in any of these attempts. These tools use different job descriptions and do not communicate with each other: putting an end to this separation process would need high efforts by all parties, therefore I need to look for different approaches.

4.3.2 Related Grid Interoperability efforts by portals

The second approach means providing a higher level tool that supports different middleware services, including job submission, brokering or storage access. Related solutions of this approach are Grid portals. The widespread and well known ones are Pegasus [76], GridFlow [16], K-Wf [62] Grid portal and Single Point of Access (SPA) portal of the HPC-Europa Project [109]. Though the first three examples provide high-level access to Grid services, they usually operate only on one middleware.

The HPC-Europa project aimed at building a Grid portal that provides a uniform and intuitive user interface to access and use resources from different domains,

so-called centers. Since most of the HPC centers have already deployed their own site-specific HPC and Grid infrastructure, it is an important requirement for them to keep the autonomy of these centers by allowing them to use their middleware and local policies. There are five different systems that provide job submission and basic monitoring functionality in the HPC-Europa infrastructure: eNANOS [71], GRIA middleware [101], Grid Resource Management System (GRMS) [53], Job Scheduling Hierarchically (JOSH) [112] and UNICORE [143]. The Single Point of Access (SPA) effort of HPC-Europa provides two sets of interfaces to application users. The first one is a generic interface set that can be used by all users for most of their batch applications. These uniform interfaces are used to access the most relevant Grid functionalities, which have been identified by analysing the requirements of the centers. These key functionalities are: job submission, job monitoring, resource information access, accounting, authorization, and data management. The second, more application-specific set of interfaces, allow users to manage more complex applications by building portlets. Using these interfaces the accompanying resource managers can build a plugin-based component. These interface methods need to be used by all brokers, providing the same abstract functionality; therefore during an integration the broker would also have to be modified. From the end-user perspective, a uniform Graphical User Interface is provided that is common for all systems deployed in the HPC-Europa infrastructure. When a user wants to submit a job, the user is required to choose the center to which the job has to be submitted and to specify its requirements. There is no global scheduling, the brokering is done by the user manually. To help this selection, the system can provide a description of the capabilities of the site-specific plug-ins. In this way the user gets an XML-based description of the methods the appropriate plug-in supports, and a description of the data structures to be used for invocation (e.g., job description).

My proposed multi-grid brokering solution is provided by the P-GRADE Portal [46], which is a Grid portal, with the main goal to support all stages of multi-grid workflow development and execution processes in various production Grids. It enables graphical design of workflows created from various types of executable components (sequential and parallel), executing these workflows in Globus-based computational Grids relying on user credentials, and finally, analysing the monitored trace-data by the built-in visualization facilities. The following functionalities are supported: defining Grid environments, creation and modification of workflow applications, managing Grid certificates, controlling the execution of workflow applications on Grid resources

and monitoring and visualizing the progress of workflows and their component jobs. The portal is interfaced to different brokers to access different VOs and Grids, such as LCG-2, gLite WMS [92], GTbroker and the NorduGrid broker [122]. During the workflow development process, the user can choose one from the interconnected production Grids. Furthermore resource managers or specific resources can also be selected afterwards. This manual selection need to be done by the user (just like in the previous HPC-Europa approach). To help the users, this portal provides GUI for resource information and a specific workflow for VO-usability test.

In both of these interoperable portal approaches users can submit jobs to different domains in a transparent way. Though this provides some level of interoperability, the users still need to be aware of the capabilities of the available resource managers, and they need to gather or track resource information themselves. In order to automate these tasks higher level approaches are needed, which are discussed next.

4.3.3 Related Grid Interoperability efforts with higher level Grid resource management

Meta-brokering means a higher level solution that schedules user jobs among various Grid brokers/domains. One of these meta-brokering approaches aims at enabling communication among existing resource brokers. The GSA-RG of OGF [124] works on a project for enabling Grid scheduler interaction. They try to define common protocol and interface among schedulers enabling inter-Grid usage. Though a common interface for inter-broker communication would enhance interoperation of different Grids, it usually takes a long time to standardize such protocols. To achieve this, they use standard tools (JSDL [113], OGSA [123], WS-Agreement [89], and propose an SDL (Scheduling Description Language) to extend the currently available job description language. Lately researchers of this group were paying more attention to agreements, and examined how to use WS-Agreement to make negotiations and perform interaction [75].

Other groups have started to develop their own protocols for inter-broker communication. One of these groups is the Latin American Grid initiative (LA Grid [72, 116]), which is a multifaceted international academic and industry partnership between major institutions in the United States, Mexico, Argentina, Spain and other locations around the world. The LA Grid main research areas are transparent Grid access, autonomic resource management and job flow management. The

meta-scheduling project in LA Grid aims to support Grid applications with resources located and managed in different domains. They define broker instances with a set of functional modules: connection management, resource management, job management and notification management. These modules implement protocols used in LA Grid through web services. A broker instance interacts with existent brokers within the resource domain. Each broker instance collects resource information from its neighbours and saves the information in its resource repository. The resource information is distributed in the Grid and each instance will have a view of all resources. The resource information is in aggregated forms to save storage space and communication bandwidth. A job request can be submitted to any known broker instance using web services. When a job request arrives, the broker matches the job to a domain with the appropriate set of resources. The matching algorithm is influenced by multiple factors; an important one is the location of resources such that the preference will be given to the local domain in which the job is submitted. If the matched resources are outside of the domain, the job is routed to a broker instance in another domain. From that point this additional broker instance is responsible for dispatching the job again, if needed, and reporting the job states back to the instance where the job was originally submitted.

The Koala Grid scheduler [115] was designed to work on DAS-2 interacting with Globus [99] middleware services with the main features of data and processor co-allocation, and it has been extended to support DAS-3 and Grid'5000. To interconnect different Grids, they have also decided to use inter-broker communication. Their policy is to use a remote Grid only if the local one is saturated. They use a so-called delegated matchmaking (DMM) [40], where Koala instances delegate resource information in a peer-2-peer manner. The LA Grid approach is similar, but they share aggregated resource data, while DMM uses a ranking of domains by their resources. For preliminary test they have built a simulator that behaves similarly as the previously mentioned Grids. The simulations results show that their architecture accommodates equally well for low and high system loads.

Gridway [105] has also broadened its support for multiple Grids. They introduce a Scheduling Architectures Taxonomy (SAT) [55] where they describe a Multiple Grid Infrastructure. It consists of different categories, we are interested in the Multiple Meta-Scheduler Layers, where Gridway instances can communicate and interact through Grid gateways (these instances are called GridGateways). These instances can access resources belonging to different administrative domains (Grids/VOs). The

basic idea is to pass user requests to another domain, when the current is overloaded – this approach follows the same idea as the previously introduced Koala DMM. Gridway is also based on Globus, and has been extended for GT4 [99] and gLite [100]. They further developed this idea and in their latest work they demonstrate interoperation with Gridway on federation of Grids [85] (namely on TeraGrid, EGEE and OSG).

The InterGrid approach [4], which promotes interlinking of Grid systems through peering agreements to enable inter-Grid resource sharing is similar to the idea of the federation of Grids. This approach has an economical view, where business application support and sustainability are primal goals. In this architecture, so-called IntraGrid Resource Managers (IRM) would play the role of Resource Brokers. The InterGrid Gateway (IGG) would be responsible of making agreements with other Grids through their IRMs. Their main tasks are to manage peering agreements and discover and allocate resources from different domains. The disadvantages are also similar: all IRMs should use the same protocol, therefore existing brokers should be redesigned to participate in this system.

Comparing the previous approaches, I can state that all of them use a new method to expand current Grid resource management boundaries. Meta-brokering appears in a sense that different domains are being examined as a whole, but they rather delegate resource information among domains, broker instances or gateways through their own, implementation-dependent interfaces. Usually the local domain has preference, and when a job is passed to somewhere else, the result should be transferred back to the initial point. Regarding multi-grid usage, the existing Grids are very strict and conservative in the sense that they are *very reluctant* to introduce any modification that is coming from research or from other Grid initiatives. Hence these solutions aiming at inter-connecting the existing brokers through common interfaces require a long standardization procedure before it will be accepted and adapted by the various Grid communities. The idea of a Grid federation based on GridWay is the most advanced solution, its only disadvantage that it fails to unify different user communities relying on brokers offering special services we have seen in the taxonomy (eg. co-allocation, checkpointing and so on).

On the other hand the advantage of my proposed meta-brokering concept is that it supports *highest number of broker capabilities* possible, and it does not require any modification of the existing Grid resource managers, since it utilizes and delegates broker information by reaching them through their current interfaces. An important

characteristics of Grid Interoperability is to support all the special user requirements that are available (see the taxonomy categories in Section 2.2).

4.3.4 Classification of related works

Now we have the comparative descriptions of the related works and approaches, I continue with the formal classification based on the ASM model introduced in Section 3.3 of Chapter 3.

As we have seen above, four solutions fall into the Broker extension group. Gridbus is able to submit jobs to Globus, UNICORE and Nordugrid – let us denote these Grids by $g, u, n \in GRID$ respectively. Gridway can be used in Globus, gLite, Nordugrid and OSG – similarly denoted by $g, l, n, o \in GRID$. JSS can utilize Globus and Nordugrid resources ($g, n \in GRID$), while GTbroker can use Globus, LCG-2 and gLite resources ($g, e, l \in GRID$). Therefore the following statement holds for the brokers (denoted by $b_{name} \in BROKER$) in this group:

$\exists h_e, h_g, h_l, h_n, h_o, h_u \in HOST:$
$\text{provides}(g, h_g) = true$ &
$\text{manages}(\ b_{Gridbus}, h_g) = true$ & $\text{manages}(\ b_{Gridway}, h_g) = true$ &
$\text{manages}(\ b_{JSS}, h_g) = true$ & $\text{manages}(\ b_{GTbroker}, h_g) = true$

$\text{provides}(n, h_n) = true$ & $\text{manages}(\ b_{Gridbus}, h_n) = true$ &
$\text{manages}(\ b_{Gridway}, h_n) = true$ & $\text{manages}(\ b_{JSS}, h_n) = true$

$\text{provides}(l, h_l) = true$ &
$\text{manages}(\ b_{Gridway}, h_l) = true$ & $\text{manages}(\ b_{GTbroker}, h_l) = true$

$\text{provides}(u, h_u) = true$ & $\text{manages}(\ b_{Gridbus}, h_u) = true$

$\text{provides}(o, h_o) = true$ & $\text{manages}(\ b_{Gridway}, h_o) = true$

$\text{provides}(e, h_e) = true$ & $\text{manages}(\ b_{GTbroker}, h_e) = true$

The SPA of HPC-Europa and the P-GRADE portal fall into the multi-brokering group. SPA can utilize eNanos, GRMS and JOSH, so the following statement holds

for it:

$\forall r_i \in REQUIREMENT, 1 \leq i \leq k : \exists h \in HOST, \exists pr \in PRESOURCE :$
belongsTo$(pr, h) = true$ & compatible(attr(r_i), attr$(pr)) = true$ &
(manages($b_{eNanos}, h) = true$ OR manages($b_{GRMS}, h) = true$ OR
manages($b_{JOSH}, h) = true$)
and
$\forall r_j \in REQUIREMENT, 1 \leq j \leq m : \exists p \in PROPERTY :$
compatible(attr(r_j), attr$(p)) = true$ &
(have($b_{eNanos}, p) = true$ OR have($b_{GRMS}, p) = true$ OR
have($b_{JOSH}, p) = true$)

The P-GRADE portal manages GTbroker, the Nordugrid broker and WMS of EGEE, therefore it is true that:

$\forall r_l \in REQUIREMENT, 1 \leq l \leq n : \exists h \in HOST, \exists pr \in PRESOURCE :$
belongsTo$(pr, h) = true$ & compatible(attr(r_l), attr$(pr)) = true$ &
(manages($b_{GTbroker}, h) = true$ OR manages($b_{Nordugrid}, h) = true$ OR
manages($b_{WMS}, h) = true$)
and
$\forall r_q \in REQUIREMENT, 1 \leq q \leq t : \exists p \in PROPERTY :$
compatible(attr(r_q), attr$(p)) = true$ &
(have($b_{GTbroker}, p) = true$ OR have($b_{Nordugrid}, p) = true$ OR
have($b_{WMS}, p) = true$)

As a result the SPA of HPC-Europa may satisfy $k+m$ user requirements by three brokers, and the P-GRADE portal may satisfy $n + t$ requirements by three other brokers. But users need to select manually the proper broker for all the jobs to be executed.

Regarding solutions in the third, Meta-brokering group, the first four solutions use the same broker instances in a peer-to-peer meta-brokering environment, therefore they can only provide their own broker properties:

$\forall r_i \in REQUIREMENT, 1 \leq i \leq k : \exists h \in HOST, \exists pr \in PRESOURCE :$
belongsTo$(pr, h) = true$ & compatible(attr(r_i), attr$(pr)) = true$ &

manages(b_{LA}, h) = $true$
and
$\forall r_j \in REQUIREMENT, 1 \leq j \leq m : \exists p \in PROPERTY$:
compatible(attr(r_j), attr(p)) = $true$ & have(b_{LA}, p) = $true$

The same holds for b_{KOALA}, $b_{InterGrid}$ and $b_{GridwayFederation}$. While in case of GMBS the following statement holds for all b_z managed by GMBS:

$\forall r_l \in REQUIREMENT, 1 \leq l \leq n : \exists h \in HOST, \exists pr \in PRESOURCE$:
manages(b_z, h) = $true$ & belongsTo(pr, h) = $true$ &
compatible(attr(r_l), attr(pr)) = $true$
and
$\forall r_q \in REQUIREMENT, 1 \leq q \leq t : \exists p \in PROPERTY$:
compatible(attr(r_q), attr(p)) = $true$ & have(b_z, p) = $true$

Since GMBS can utilize all brokers that can be described by BPDL (at least the 16 brokers of the taxonomy) and therefore it represents all their properties, it is true that $m << t$, so it can provide the *highest level* of Grid Interoperability with automatic broker selection.

The classification of the previously introduced solutions according to the formal statements given above can bee seen in Table 4.3. A bullet in the appropriate column means that the actual solution is able to provide the level of interoperability the column denotes. "M" means manual and "A" means automatic selection from the available Grids (or brokers) for a user request.

4.4 Outlook of GMBS

Even though GMBS has reached a maturity state in its present form that it is able to solve the problem of Grid Interoperability, development does not stop here. There are several *ongoing collaborations* that work on extensions of GMBS in order to prepare it to cope with future challenges.

Since currently user communities are tightly coupled to brokers or Grid portals, the best way to use GMBS is a *matchmaking service* for Grid portals. Together with my colleagues we are currently working on interfacing GMBS with the P-GRADE Portal [46] and the WS-PGRADE portal [43]. Recent Grid usage trends show that

Table 4.3: Classification of Grid Interoperability solutions.

Solution	**Low-level interoperability**		**High-level interoperability**	
	M	**A**	**M**	**A**
Broker extension				
Gridbus	•			
Gridway	•			
GTbroker	•			
JSS	•			
Multi-brokering				
HPC-Europa	•	•	•	
P-GRADE Portal	•	•	•	
Meta-brokering				
LA Grid	•	•		
Koala DMM	•	•		
InterGrid	•	•		
Gridway Federation	•	•		
GMBS	•	•	•	•

new or growing user communities set up a new portal to serve the community instead of joining portals of other user communities (see the growing number of portal installations [132] in case of the P-GRADE portal). Till this trend continues, all portals will have their own GMBS instance that will be able to serve their users. This means that the centralized architecture of GMBS *will not become* a bottleneck for job submissions in the portal. Nevertheless we cannot say that such a situation will not ever appear in the future. To target this scalability issue, we have already started a collaboration with the Technical University of Delft for investigating *load sharing* with multiple meta-broker instances connected to a peer-to-peer network [49]. This solution will avoid possible future bottlenecks and will be capable of serving thousands of users accessing a single GMBS instance.

In an other collaboration we have investigated the scheduling part of the GMBS matchmaking process, and designed a Decision Maker component in [P13] as an *extension* of the MatchMaker component of GMBS. The first part of the matchmaking remained unchanged: the list of the available brokers is filtered according to the requirements of the actual job read from its job description. Then the list of the remaining brokers along with their performance data and background load are sent to the Decision Maker in order to determine the fittest broker for the actual job. The

Decision Maker uses a *random number generator*, which generates pseudo-random numbers. We also developed a unique random number generator, which generates random numbers with a given distribution, which we called a generator function. To improve the scheduling performance of the Meta-Broker we need to send the job to the broker that best fits the requirements and executes the job without failures with the shortest execution time. Every broker has three properties that the algorithm can rely on: the successful counter, the failure counter and the load counter. We have developed four different kinds of decision algorithms using these counters and shown that the scheduling process of GMBS can be improved with the Decision Maker component. Later on in [19, 20] we have developed an additional algorithm based on a weighted fitness function that uses the *Pliant System*, which also brought additional *performance gain*.

In this dissertation I particularly focused on Service Grids. *Desktop Grids* are commonly known as volunteer computing systems, because they often rely upon the general public to donate compute resources or spare cycles. Unlike Service Grids, which are based on complex architectures, volunteer computing has a simple architecture and has demonstrated the ability to integrate dispersed, heterogeneous computing resources with ease, successfully scavenging cycles from tens of thousands of idle desktop computers. These two kinds of Grid systems have been completely *separated*, hence there has not been a mechanism to exploit their individual advantageous features in a *unified environment*. However, with the objective to support new scientific communities who need extremely large numbers of resources, the solution could be to interconnect these two kinds of Grid systems into an integrated Service Grid – Desktop Grid (SG-DG) infrastructure. User communities have gathered around various Grid systems (including Service and Desktop Grids) forming separate islands that represent borders they cannot cross. As these communities are growing and demanding more and more computational power, uniting these islands draws more attention in Grid research and development. An approach to extend GMBS to Desktop Grids is given in [48]. In this paper we investigated with my colleagues, how to use the *3GBridge* service [84] acting as a Service Grid broker that can be managed by GMBS.

Cloud Computing [14] is a novel infrastructure that focuses on commercial resource provision and virtualization. Grids already provide solutions for executing complex user tasks, but they are still lacking non-functional guarantees. The newly emerging demands of Grid users and researchers call for expanding current execution models

with business-oriented utilization and support for special services. These demands could be fulfilled in Grids by using the novel virtualization technologies developed for Clouds. Looking beyond Grids, in the near future such complex applications will appear that require the simultaneous utilization of Grid, Web and Cloud services. To target this problem area, we have started a collaboration with the Technical University of Vienna to develop a unified service architecture called *SLA-based Resource Virtualization* (SSV) that builds on three main areas: agreement negotiation, brokering and service deployment using virtualization. The brokering functionalities of this architecture is provided by GMBS. The basic requirements of this architecture, the corresponding components and their interactions have been published in [P15] and [50].

4.5 Summary

In this chapter I introduced a general architecture for meta-brokering and described the development of the Grid Meta-Broker Service. I gathered the related approaches in the literature that have appeared up to now, and classified all the available interoperable solutions. This classification shows that GMBS provides the highest level of Grid Interoperability. Finally I gave an outlook where GMBS continues to evolve, and how it will serve as a general gateway for heterogeneous distributed environments beyond Grids in the future. The results of this chapter belong to theses II and III, and were published in papers [P3], [P6], [P7], [P8], [P9], [P10], [P12], [P14], [P13], [P15], [P17], and [P19].

Chapter 5

The evaluation of Grid meta-brokering

5.1 Evaluation of GMBS

5.1.1 Evaluation methodology

In Section 4.3.3 of Chapter 4 we have seen how the meta-brokering service proposed in this dissertation and the related solutions address Grid Interoperability, and we argued that GMBS provides the highest level of interoperability by satisfying most user requirements with automatic broker selection. Therefore the evaluation methodology addressed in this section targets the speedup of scientific Grid applications measured by using the proposed interoperable GMBS meta-brokering service, compared to the usual, non-interoperable case. As I mentioned in the introduction of this dissertation, various production Grids have been set up all around the world attracting separate user communities. As a result, in this general, non-interoperable Grid utilization different users submit their jobs to different production Grids they belong to. In the following experiments I use random distribution of user jobs in order to simulate this non-interoperable utilization.

In order to address universality, I use real user application execution traces as background workload gathered both in parallel and production Grid environments, published in publicly available archives [129, 107]. Since these traces contain various jobs of hundreds of users, a general job execution time cannot be determined. Therefore, I decided to rely on my own experiences of application gridification in selecting the reference Grid application for the simulations. In the beginning of Chapter 2 I also discussed that user support teams have been established in order to help applica-

tion gridification. I have been involved in the work of the Grid Application Support Centre [103], and learned how the decomposition of the original application should be selected in order to reduce the overall makespan of its jobs. According to these experiences, in the evaluation I examine how meta-brokering may speed up application executions decomposed to jobs running around 10 minutes.

Although I have gathered 16 brokers surveyed in the taxonomy in Section 2.2 of Chapter 2, not all of them are used in current production Grids by large number of users. Taking into account the major service Grids around the world, their aggregated access through meta-brokering to establish Grid Interoperability could be achieved with the integrated management of around 10 brokers. Since the usage of Service Level Agreements [65] is still rarely supported in production Grids, scientific applications use only some special requirements. They are expected to increase in the near future, but the current non-interoperable Grid systems hinders using complex requirements (which would reduce the number of suitable resources). According to these current practical usage conditions, in the following subsections I evaluate the performance of meta-brokering in simulated Grid environments managed by 6 to 14 brokers having various properties. Each broker has an own simulated Grid with 16 to 48 nodes loaded by real workload traces.

5.1.2 Grid meta-brokering simulation architecture

Regarding general purpose Grid simulations, two main solutions have been developed: SimGrid [57], GridSim [12]. For simulating Grid scheduling, additional tools have been developed: GangSim [22], GSSIM [54], OptorSim [5] and Alvio [36]. GangSim is specialized in SLA-based resource sharing simulations. GSSIM is based on GridSim and it can be used to evaluate various scheduling algorithms for Grid brokers. OptorSim is designed to test dynamic replication strategies used in optimising the efficiency of data movements in Grids. Finally Alvio provides a framework for evaluating the performance of different Grid scheduling policies. Most of these tools do not support evaluating higher level, meta-brokering algorithms in their current forms. Only an extension of the Alvio simulator deals with built-in meta-brokering policies, but it rather focuses on job forwarding among brokers in a peer-to-peer fashion.

In order to evaluate my proposed meta-brokering service, I have designed a general meta-brokering simulation environment, in which all the related Grid resource management entities can be simulated and coordinated. Since SimGrid simulates

various distributed computing platforms, it covers a wider application area, less specialized in Grid systems and has less impact in Grid research. On the other hand, the GridSim toolkit is a fully extendible, widely used and accepted Grid simulation tool – these are the main reasons why I have chosen this toolkit for my simulations. It can be used for evaluating VO-based resource allocation, workflow scheduling, and dynamic resource provisioning techniques in global Grids [12]. It supports modeling and simulation of heterogeneous Grid resources, users, applications, brokers and local schedulers in a Grid Computing environment. It provides primitives for the creation of jobs (called gridlets), mapping of these jobs to resources, and their management, therefore resource brokers can be simulated to study scheduling algorithms. It provides a multilayered design architecture based on SimJava [37], a general purpose discrete-event simulation package implemented in Java. All components in GridSim communicate with each other through message passing operations defined by SimJava.

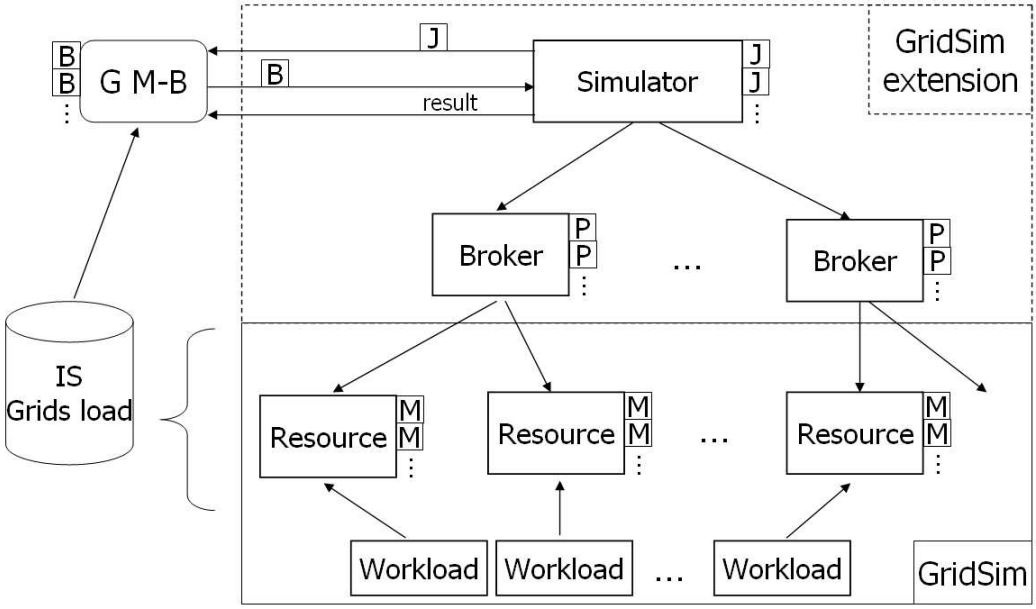

Figure 5.1: Meta-brokering simulation environment.

The *general meta-brokering simulation architecture* can be seen in Figure 5.1. On the right-bottom part we can see the GridSim components used for the simulated Grid systems. Resources can be defined with different Grid-types. Resources consist of more machines, to which workloads can be set. As an extension of GridSim classes, I have developed the Broker and Simulator entities in order to enable the simulation

of meta-brokering. On top of this simulated Grid infrastructure we can use the Broker entities for setting up brokers with various scheduling policies, while the Simulator component is responsible for parametrizing and executing each experiment. Before the GMBS is used in the simulation, it has to be configured with BPDL descriptions, and the job requests need to be submitted in JSDL.

Brokers are extended GridUser entities:

- they can be connected to one or more resources;
- different properties can be set to these brokers (agreement handling, co-allocation, advance reservation, etc.);
- some properties can be marked as unreliable;
- various scheduling policies can be defined (pre-defined ones: *rnd* – random resource selection, *fcpu* – resources having more free CPUs or less waiting jobs are selected, *nfailed* – resources having less machine failures are selected);
- generally resubmission is used, when a job fails due to resource failure;
- finally they report to the IS (Information System) Grid load database.

The Simulator is an extended GridSim entity:

- it can generate a requested number of gridlets (jobs) with different properties, start and run time (length);
- it is connected to the created brokers and able to submit jobs to them;
- the default job distribution is the random broker selection (though at least the middleware types are taken into account);
- in case of job failures a different broker is selected for the actual job;
- it is also connected to the Grid Meta-Broker Service through its web service interface and able to call its matchmaking service for broker selection.

Table 5.1: Evaluation results of the first experiment.

Brokers	Resources	Jobs	Work-load	AVG Time RND	AVG Time MB
3/X – 3/Y (*rnd*)	6x4(x4)	100	20x(6x4)	1086140.78	93959.86 **8.6%**
3/X – 3/Y (*fcpu*)	6x4(x4)	100	20x(6x4)	255944.12	18469.28 **7.2%**

5.1.3 Evaluation with parallel workloads

For the evaluation of the GMBS I executed three experiments with parallel workloads, each of them is summarized in Table 5.1, 5.2 and 5.3 respectively. The tables have six columns, each row denotes a simulation with the specified environment setups. The first column shows the number of brokers participating in the actual simulation, their middleware types and the used scheduling policy (e.g. 3/X - 3/Y (*rnd*) means that 6 brokers are used in the simulation: three of them are operating on middleware X, the remaining three are using middleware Y, and all brokers use the *rnd* policy). The second column shows the number of resources participating in the simulation: 6x4(x4) denotes an environment consisting of 6 brokers operating on 4 resources each; each resource is a cluster of 4 machines. The third column shows the number of jobs submitted in a run of the simulation. The fourth column shows the number of jobs submitted to each resource as background load: 20x(6x4) means that 20 jobs were submitted to each cluster of the 6 brokers, i.e. 480 workload jobs have been submitted to 24 clusters during the simulation. I used the cleaned SDSC Blue Horizon workload logs from the Parallel Workloads Archive [129], which contain detailed workload logs collected from large scale parallel systems in production use in various places around the world. This cleaned log file contains data on 243,314 jobs of 468 users with 32 months of activity. I have chosen this trace, because it represents a high variety of user applications submitted by many users. This log had the longest time frame with the highest load from the available traces in the archive. These were important characteristics, because I had to partition these logs to feed simulated resources of several simulated Grids/VOs. I have analysed the traces in order to select representative fragments, and further partitioned this file into separate workload files to feed the clusters of the simulated Grids as background load. In case of the setup denoted by 20x(6x4), I created 24 workload files, each of them contained 20 jobs. The sorting of the jobs to files from the original log file has been done in a continuous manner, and

their arrival times have been manually rescaled to fit the simulation time interval; the runtime of the jobs remained the same as in the original log. The fifth column shows the average simulated run time of the jobs, when I used random distribution among the brokers. The sixth column shows the average simulated run time for the jobs submitted to brokers selected by the GMBS, and also denotes the percentage according to the measured times in the fifth column.

Table 5.2: Evaluation results of the second experiment.

Brokers	**Resources**	**Jobs**	**Workload**	**AVG Time RND**	**AVG Time MB**
8/X (*rnd*)	8x4(x4)	100	20x(8x4)	1320141.79	226344.34 **17.1%** 2nd: 22304.95 **1.7%**
8/X (*fcpu*)	8x4(x4)	100	20x(8x4)	236322.30	117563.16 **49.7%** 2nd: 14318.0 **6.1%**

In the *first* experiment, which is summarized in Table 5.1, I submitted 100 jobs at a time. I used an environment consisting of 6 brokers operating on four resources each; each resource had four machines. Three brokers used resources with GRID_X middleware and the other three used resources with GRID_Y middleware. Four special properties (checkpointing, advance reservation, co-allocation and agreement handling) were distributed among the brokers, each broker had two special properties, out of which one was unreliable (50% failure). Half of the jobs were sent to GRID_X, the rest to GRID_Y. 20% of the jobs had no special property, the rest of the jobs had one special property and all the four properties were distributed equally among the jobs (20 jobs had no special property, 20 jobs had checkpointing requirement, 20 jobs required co-allocation, 20 jobs had advance reservation requirement and the rest 20 jobs required agreement handling). The run time of the jobs took around 5 minutes each. The first row denotes a configuration, in which 6 brokers used random selection policy (which is the first one of the available policies described in subsection 5.1.2), 100 jobs were submitted into the system and 20 jobs were submitted to each resource (cluster) as background workload. In the second row the *fcpu* policy is used by the brokers, the number of jobs and workload samples were the same.

Due to the broker property distribution in the first simulation setup, a job with a

Table 5.3: Evaluation results of the third experiment.

Brokers	Resources	Jobs	Workload	AVG Time RND	AVG Time MB
6/X (*fcpu*)	6x8(x2)	100	20x(6x8)	804538.18	286968.93 **35.7%**
3/X (*nfail*)	3x10(x2)		20x(3x10)		With training:
1/X (*rnd*)	1x16(x2)		20x(1x16)		52990.76 **6.6%**
6/X (*fcpu*)	6x8(x2)	1000	50x(6x8)	2268272.70	737762.46 **32.5%**
3/X (*nfail*)	3x10(x2)		50x(3x10)		With training:
1/X (*rnd*)	1x16(x2)		50x(1x16)		373857.80 **16.5%**

special property running on either middleware could surely successfully run only on one specific broker. This caused overloading of some brokers even with the use of the GMBS, therefore I created a different environment. In the *second* experiment I set up 8 brokers operating on 4 resources each (just like in the previous configuration), but all having the same GRID_X middleware. The same property distribution was used for the jobs. The brokers had 2-2 special properties again, but every second broker had one unreliable property (in this case two brokers could run some job the same time without any failures). The results are shown in Table 5.2. The last column contains additional information: this means I repeated the measurement again, in a way that the brokers were aware of previous submission failures.

5.1.4 Evaluation with preliminary training

In the second experiment I realized that repeating measurements on the same environment setup caused additional performance gain. In such cases the first evaluation run can be regarded as a training phase, in which the meta-broker learns which properties of the brokers are unreliable. Based on these experiences, I developed a preliminary training phase, in which I submit training jobs with different property requirements to each broker. As a result of these submissions, the dynamic performance data of the participating brokers are initiated in the BPDLs. In the following experiments I also measured the performance results of runs using such training phases.

In the *third* experiment (shown in Table 5.3) I submitted first 100, then 1000

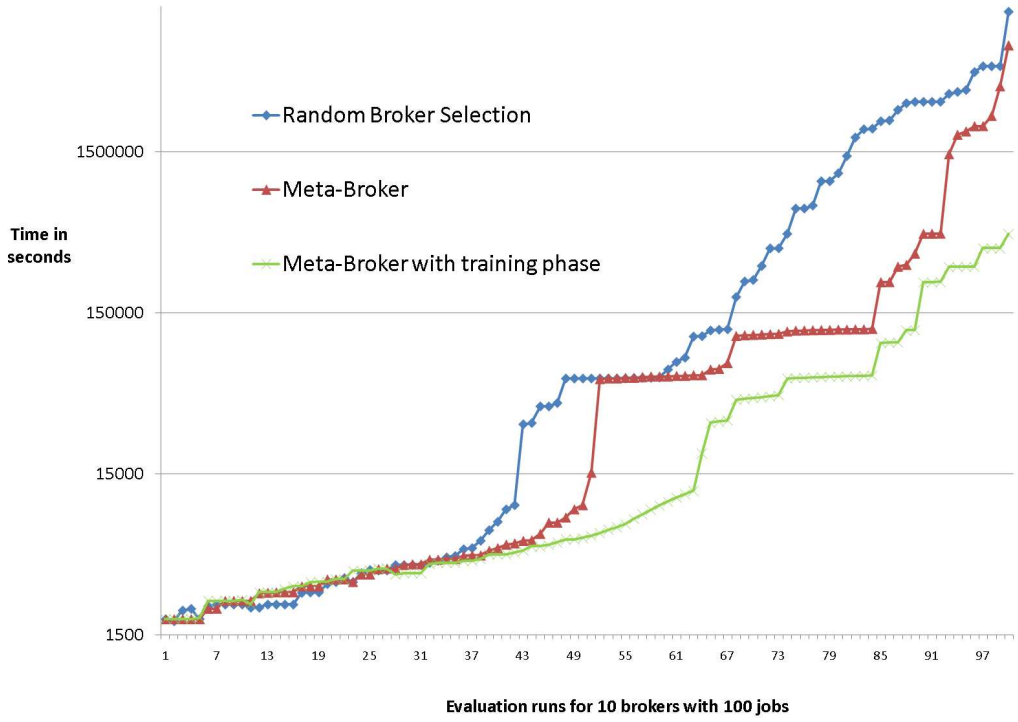

Figure 5.2: Evaluation diagram corresponding to the first row of Table 5.3

jobs at a time. I defined 10 brokers running on Grids with similar middleware. I distributed three properties (checkpointing, co-allocation and agreement handling): 6 brokers had 1-1 property, out of which three brokers were unreliable. These 6 brokers were running on 8 resources each. Three other brokers had 2-2 properties, 1-1 were unreliable. These three brokers were running on 10 resources each. Finally, the 10th broker had no special property and ran on 16 resources. 40% of the jobs had no special property, the rest were distributed equally among the three properties. The run time of the submitted jobs was around 10 minutes. In the training phase 10 preliminary jobs have been submitted in order to initiate the BPDLs. 50 jobs were submitted to each resource by the workload entities in the second phase of this experiment, which put heavier load on the resources. The last column of Table 5.3 shows the results of these simulations.

Figure 5.2 and 5.3 depict detailed evaluation runs with logarithmic time curves for randomized broker selection, Meta-Broker utilization and enhanced Meta-Broker utilization with preliminary training phases. The detailed values correspond to the first and second rows of Table 5.3, respectively. In Figure 5.4, we can see the summarized results denoting average runtime values for each experiment setup, which

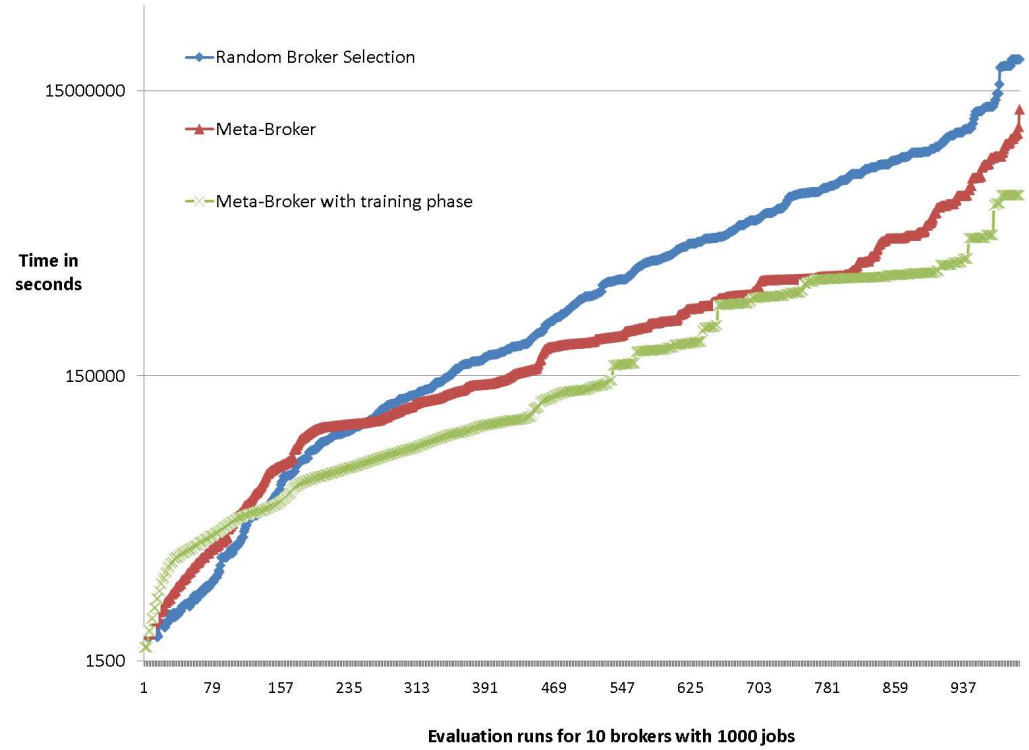

Figure 5.3: Evaluation diagram corresponding to the second row of Table 5.3

clearly show that the Grid Meta-Broker Service provides less total execution time (makespan) by automating broker and Grid selection for users. During utilization it is able to adapt to broker failures and to avoid selecting overloaded Grids. Using a preliminary training phase can significantly improve the performance.

Taking a look at these three simulation experiments we can see that the environments are reasonably chosen. For the first time I set up a simple environment with 3-3 brokers on different Grid middleware. For the second time I omitted differentiating the middleware types, because in this environment it meant only another broker property (it did not require additional scheduling steps). I also enlarged the environment by two more brokers up to 8, and set 4 brokers totally reliable. For the third time I scaled the environment up to 10 diverse brokers operating on clusters with different sizes. Finally I have doubled the execution time of the jobs from 5 to 10 minutes, scaled the number of submitted job up to 1000 and the workload jobs to 50 per cluster within the same environment. I can conclude that meta-brokering caused significant speedup in all the presented experiments.

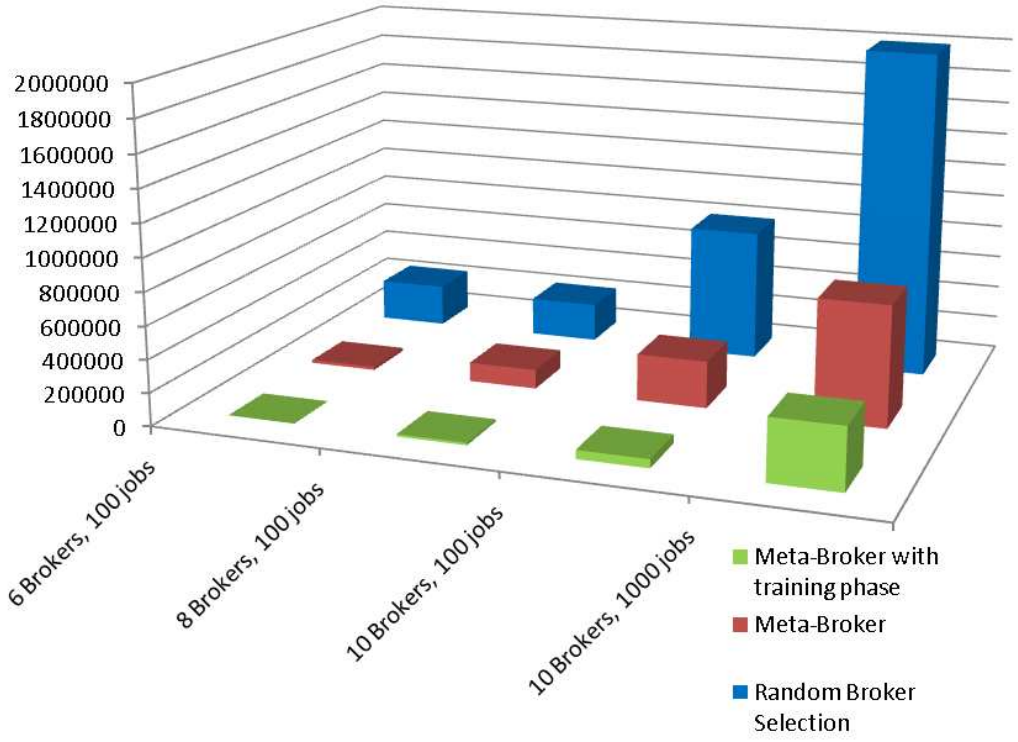

Figure 5.4: Evaluation results of runs in the first three experiments

5.1.5 Evaluation with Grid workloads

Though in the previous subsection I have measured convincing utilization gains by using the GMBS for job submissions to different simulated Grid environments based on traces of parallel environments, I decided to create a larger simulation environment closer to the latest Grid usage trends. This simulation setup was derived from real-life *production Grids*. Since current Grids and brokers support only a few special properties, I used four in these simulations. To determine the (proportional) number of resources in the simulated Grids I compared the sizes of current production Grids (EGEE VOs, DAS3, NGS, Grid5000, OSG, ...).

Table 5.4 shows the evaluation environment used in this *fourth* experiment, and the average of the measured evaluation runs. I used similar notations in this table as in the previous subsection. In this evaluation I utilized 14 brokers, all used the (best performing) *fcpu* scheduling policy – this is denoted by the first column of Table 5.4. The second column shows the number of resources connected to the brokers (I used the same notation as in the previous tables). In this case I submitted 1000 jobs to the system (see third column of Table 5.4) after a delay of 6000 simulated seconds

(in order to let the workload jobs started in this initial warm-up period and to avoid measuring unrepresentative data), and measured the makespan of all the jobs. An application of 1000 jobs can be so complex that it may easily overload a particular Grid, therefore multiple Grid utilization is definitely needed. The run time of the jobs were set to 10 minutes. Out of the 1000 jobs 100 had no special property, and for the rest of the jobs the four properties were distributed in the following way: 30 jobs had agreement handling property, 30 had advance reservation, 20 had co-allocation and 10 had checkpointing as requirement (note that the actual realization of these properties is irrelevant for the simulated evaluation). The properties were distributed among the 14 brokers: 9 brokers had only one property, out of which 3 were unreliable, 4 brokers had two properties each, out of which 3 were unreliable, and finally one broker had three properties with one unreliable property (unreliability means 50% failure).

Table 5.4: Evaluation results of the fourth experiment.

Brokers	**Resources**	**Jobs**	**AVG Time RND**	**AVG Time MB**	**AVG Time MB with training**
14/X (*fcpu*)	2x4(x4) 2x6(x4) 3x8(x4) 4x10(x4) 3x12(x4)	1000	68583.03	34225.35 **49.9%**	33488.21 **48.8%**

The workload log was selected from the Grid Workloads Archive (GWA [107, 41]). I used the GWA-T-11 LCG Grid trace file. This log contains 11 days of activity with 188,041 jobs of 216 users sent to 170 nodes that made up the LCG Grid [92] in 2005. The main reasons for choosing this trace were that it contained the highest number of processors, it had a time frame that fits the execution of an application of 1000 jobs, and its jobs were categorized according to the nodes they were submitted to. In this revised simulation architecture I used 120 nodes (called resources in the simulation), therefore I partitioned the logs and created 120 workload files (out of the possible 170 nodes included in the log). At this time I left the number of jobs per node, and the arrival and execution times untouched. These files were fed to the simulation environment as background workload. In addition to the table description seen in subsections 5.1.3 and 5.1.4, Table 5.4 contains a separate column denoting the results of GMBS utilization including training phases. This time 100 jobs were submitted to each broker prior the evaluation runs to initiate the performance values.

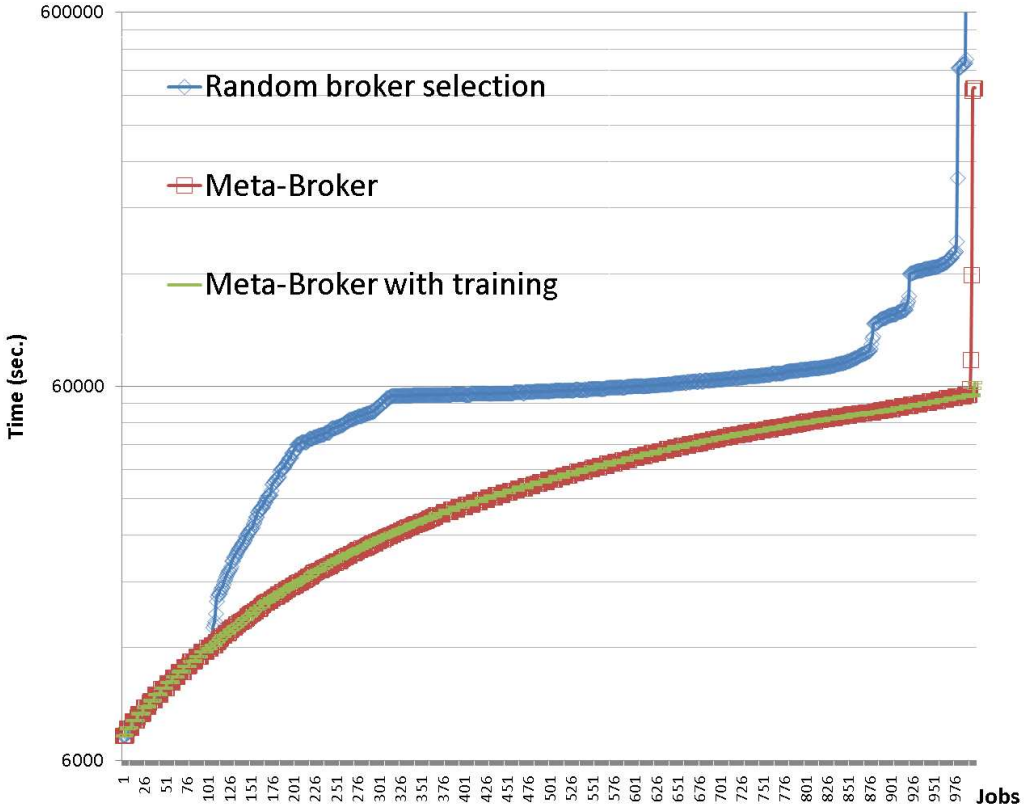

Figure 5.5: Compared evaluation results for the three types of runs

For this *fourth* experiment I repeated the simulations five times for each setup, and compared the averages of the evaluation measurements. The results are shown in Figure 5.6. The last columns represent the average values of the five runs (Avg), which are also shown in the last three columns of Table 5.4. In order to compare the three types of simulation runs I took the ones closest to the average and depicted them in Figure 5.5. Here we can see that after 100 jobs the random selection picked heavily loaded or unreliable brokers (as an ordinary user would behave), while submissions with the GMBS *avoided* utilizing these brokers. Due to uninitialized BPDLs and the *heavy load* caused by the workload jobs, there were still some submissions with the GMBS that took more then 60000 simulated seconds, which were successfully eliminated in another evaluation run of GMBS using the preliminary training phase. The results show that in a simulated Grid environment with real Grid workload, under a high number of job submissions the GMBS utilization was able to *shorten the makespan* by more than 50%. This time the training phase gave only a little gain, because the GMBS was able to adapt to broker failures soon due to the high number

of submitted jobs.

Summarizing the evaluation results of the four experiments I can state that the interoperable meta-brokering solution of GMBS achieved *much better (in some cases an order of magnitude better) performance* in Grid application execution compared to the general, non-interoperable Grid utilization.

Regarding the overhead the meta-brokering layer generates, we only need to consider the latency of the web service calls and the matchmaking time of the GMBS. In these evaluations this latency took up around 200 milliseconds for a job, so it is negligible comparing to general broker response times (which can last up to several minutes).

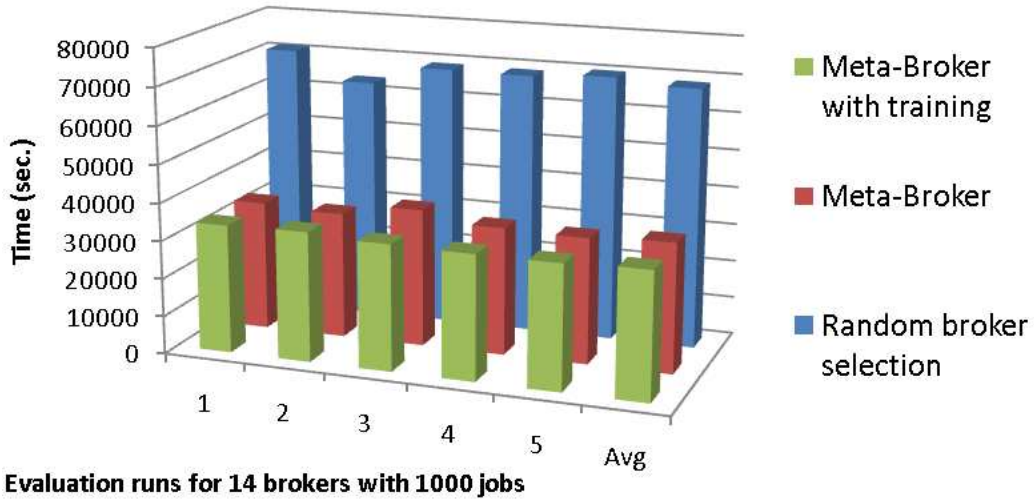

Figure 5.6: Evaluation results of runs in the fourth experiment

5.2 Summary

In this chapter I introduced a simulation architecture for meta-brokering that I used to evaluate GMBS using real parallel and Grid workloads. The performance analysis clearly showed that GMBS performs much better related to random broker selection. Furthermore, I measured approximately an order of magnitude better performance using the preliminary training phase in the experiments. The results of this chapter belong to thesis IV, and were published in papers [P17] and [P19].

Chapter 6

Conclusions

Current Grid systems are used by a high number of various research communities, but the lack of interoperability among them represents borders for further development and efficient usage. Numerous user applications are so large and complex that their executions require more computing resources than a particular Grid can provide. In order to solve this problem, I proposed in this dissertation a novel meta-brokering solution that is able to serve complex user requirements by providing transparent access to resources of several Grid systems simultaneously, in an automated way. The goal of this work was to enable Grid Interoperability by providing the highest number of brokering capabilities in a way that it does not require any changes to the underlying Grid middleware services.

The initial steps of the dissertation aimed at clarifying the roles and relations of Grid resource manager components by presenting a survey of available tools, a taxonomy of their brokering services and properties and an anatomy of their conformation. The revealed Grid brokering mechanisms are formalized by using Abstract State Machines (ASM) in order to give precise definitions including classification levels for interoperability that are used later for literature classification.

A new interoperable meta-brokering approach is introduced together with a general, abstract architecture. To enable the integrated management of different brokers, a new broker description language has been designed to describe all the available Grid resource brokers. As a proof of the concept, the components of the Grid Meta-Broker Service have been developed that perform user interactions, monitoring of resource and Grid load, tracking broker performance and automatic broker selection.

In order to evaluate the implemented meta-brokering service, a meta-brokering

simulation environment has been developed. The presented performance evaluation uses real parallel and Grid workload traces, its results affirm that the proposed meta-brokering solution enables better adaptation and achieves an order of magnitude better performance over random broker selection. Possible directions for future work are to broaden the interoperability beyond service Grids, and to extend meta-brokering to manage arbitrary heterogeneous distributed systems.

The brokering-related services of the P-GRADE portal [46] and the gUSE/WS-PGRADE system [43] are based on the contributions of this dissertation. Several scientific projects use or are supported by these systems. Therefore the results of this dissertation are used in the following European Union projects: SHIWA project [138], EDGI project [91], CancerGrid project [102] and GASuC project [103], and in the following national projects: UK ProSim project [134], MoSGrid project [118], and a biology project of ETH Zurich [141].

The scientific results of the theses have been published in numerous journals, conference and workshop papers and have been presented in various scientific forums. These publications have inspired further research, generated collaborations, and are well represented by many independent citations. Most of the research presented in this dissertation has resulted from active involvement in the CoreGRID and S-CUBE EU Network of Excellence projects [95, 137].

Appendix A

Summary in English

Introduction

Grid Computing [29] has become a separate research field in the '90s and since then it has been targeted by many projects all around the world. Several years ago users and companies having computation and data intensive applications looked sceptical at the forerunners of Grid solutions that promised less execution time and easy-to-use application development environments by creating a new virtually unified high performance system of interconnected computers from all around the world. Research groups were forming around specific parts of Grid systems and different research areas emerged, because former techniques of distributed computing were not applicable in Grid systems. Many user groups from various research fields (biology, chemistry, physics, etc.) put their trust in Grids and today usage statistics and research results show that they were undoubtedly right. Grid Computing has been in the spotlight, several international projects have aimed to establish sustainable Grids (eg. CoreGRID [95], EGEE [92], NextGRID [121], GEANT [97], KnowARC [104], EUAsiaGrid [94] and OSG [127]).

Core Grid services are provided and implemented by a so-called Grid middleware [33]. The first widespread middleware was the Globus Toolkit [30], which became a de facto standard for Grid Computing around 2002. Since then several middleware solutions have appeared, and the production Grids using these solutions formed separate islands that represent borders for both researchers and user communities. A decade of Grid development has established many national and international production Grids based on different middleware solutions (eg. HunGrid [110], NGS [120], EGEE [92],

UNICORE [143], NorduGrid [122] and OSG [127]). As a result of the numerous Grid projects and available production Grids, user support centers [142, 98, 146, 103] have been set up in order to ease application porting to Grid environments. In some cases these applications are so large and complex that their executions require more computing resources than a particular Grid can provide. Therefore similarly to the World-Wide Web, the interconnection of these separate islands can result in a World-Wide Grid in the future. Such an aggregated system could cope with the growing number of users and computation-intensive applications.

Resource management in Grid systems is the research field most affected by user demands. Though well-designed, evaluated and widely used resource managers (also called as brokers) have been developed, new capabilities are required, such as interoperability and agreement support. The available resource managers have already been surveyed by other research groups [52], but these publications do not detail capabilities related to interoperability and do not separate operational roles (eg. scheduling, brokering, management). This dissertation aims at providing a high-level brokering solution to establish Grid Interoperability [70], which means the bridging of different Grid infrastructures in order to allow users on one Grid to run computing jobs and exchange data with users on other Grids. The current solutions of Grid resource management will not be able to fulfil the high demands of future generation Grid systems, though several Grid resource brokers [2] have been developed supporting different Grid systems. The main problem is that most of them cannot cross the borders of separate Grid islands caused by different Grid middleware solutions, therefore they can mature as slowly as middleware solutions evolve. These newly arisen problems need to be treated by novel research approaches in order to aggregate the separated Grid islands and manage them together, because currently used Grid middleware solutions do not support real interoperation other then restricted bilateral ones.

Solving these problems is crucial for the next generation of Grids, which should spread from the academic to the business world. The advance of Grids seems to follow the way foreseen by the Next Generation Grids Expert Group, which has been established by the European Commission. In their third report [61] they have pointed out that Grid and web services are converging and envisaged hybrid services called as SOKUs (Service Oriented Knowledge Utility), which enable more flexibility, adaptability and advanced interfaces, therefore interoperability is evident and congenital in these systems. Following these expert guidelines and the latest requirements of Grid user groups, I propose in this dissertation such a high-level Grid brokering solu-

tion that enables Grid Interoperability by providing the highest number of brokering capabilities in a way that it does not require any changes to the underlying Grid middleware services.

New scientific results

During the research presented in this dissertation my first goal was to elaborate a classification of Grid resource brokers. At that time, the less than ten-year-old Grid Computing had several resource management solutions named by different expressions operating on different middleware addressing various user needs. During the preparation of the first thesis I examined the widespread Grid resource brokers used by different user communities, identified their key functionalities and properties, gathered them into a taxonomy, and classified them in a survey using the elements of the taxonomy. I analysed the connections and inner structures of the available Grid resource manager components, identified different operational roles and resolved their contradictory naming acronyms and expressions by creating an anatomy of Grid resource managers. I formalized the identified brokering roles, and inserted them into the Abstract State Machine (ASM) model of Grid systems [60]. I identified and defined interoperability levels for Grid brokering solutions and expressed them in the presented model that enables the classification of related brokering approaches. I stated the following thesis based on these results:

> **Thesis I. I designed a category framework of broker capabilities that I used to create a general taxonomy of Grid brokers. I designed an anatomy of Grid resource managers that I used to formalize Grid brokering levels based on the ASM model of Grids [60].**

Grid Interoperability [70] is a fundamental challenge of Grid Computing nowadays. The presented broker taxonomy also points out the heterogeneity in most brokering components and methods. The resource management anatomy revealed their similarities and possible interactions that paved the way for introducing a meta-level in Grid brokering to interoperate different Grid systems. Some of the surveyed brokers are capable of low-level interoperation by accessing resources of different Grids. I showed how these approaches address multi-grid brokering by broker-extension and multi-brokering from Grid portals. For a higher level of interoperability, a general broker description language is needed in order to enable the unified management of

Grid brokers. The second thesis contains the elaboration of such language based on a meta-data model, using the categories of the broker taxonomy.

> **Thesis II. I designed a new, XML-based description language called Broker Property Description Language (BPDL) that is able to describe any Grid resource broker that can be categorized in the taxonomy. A high-level brokering service can use this language for the unified management of these brokers.**

I named the novel approach that performs high-level brokering at the meta-level of Grid resource management as meta-brokering. The next, third thesis includes the description of the required components of a general meta-brokering architecture (besides the broker description language) and a realization of the abstract architecture in a meta-brokering service that does not require any modifications to the utilized brokers and Grids.

> **Thesis III. I determined the general requirements of Grid meta-brokering, and developed a general architecture based on these requirements that introduces a higher abstraction layer for enabling Grid Interoperability by the unified management of Grid brokers. Based on this general architecture, I designed the necessary components to build the Grid Meta-Broker Service (GMBS).**

The components of the realized meta-brokering service perform user interactions, monitoring of resource and Grid load, tracking broker performance and automatic broker selection. After publishing this meta-brokering approach, other research groups have also realized the need for interoperable brokering and started to develop their own solutions. I designed a classification of these solutions based on the interoperability levels introduced in Thesis I. The final part of the research was to evaluate the proposed meta-broker. The GridSim Toolkit [12] is a widely accepted and used Grid simulator that can be easily tailored to analyse Grid brokering methods. The fourth thesis presents a meta-brokering simulation architecture that extends GridSim, and the performance evaluation of the implemented meta-broker in this environment by using real world resource usage traces form the publicly available Parallel and Grid Workloads Archive [129, 107].

Thesis IV. I developed a new simulation environment based on the GridSim [12] simulator that is able to evaluate meta-brokering. I performed the evaluation of GMBS in this environment with a performance analysis using both real parallel and Grid workload traces. I proved the effectiveness of the interoperable meta-brokering service with the evaluation.

The evaluation results showed that the interoperable meta-brokering solution of GMBS was able to achieve an order of magnitude better performance in Grid application execution compared to the general, non-interoperable Grid utilization simulated by random broker selection.

Conclusions

Current Grid systems are used by a high number of various research communities, but the lack of interoperability among them represents borders for further development and efficient usage. Numerous user applications are so large and complex that the execution may require more computing resources than a particular Grid can provide. In order to solve this problem, I proposed a novel resource management solution in this dissertation that is able to serve complex user requirements by providing transparent access to resources of several Grid systems simultaneously, in an automated way.

The brokering-related services of the P-GRADE portal [46] and the gUSE/WS-PGRADE system [43] are based on the contributions of this dissertation. Several scientific projects use or are supported by these systems. Therefore the results of this dissertation are applied in the following European Union projects: SHIWA project [138], EDGI project [91], CancerGrid project [102] and GASuC project [103], and in the following national projects: UK ProSim project [134], MoSGrid project [118] and a biology project of ETH Zurich [141].

The scientific results of the theses have been published in numerous journals, conference and workshop papers and have been presented in various scientific forums. These publications have inspired further research, generated collaborations, and are well represented by many independent citations. Most of the research presented in this dissertation has resulted from active involvement in the CoreGRID and S-CUBE EU Network of Excellence projects [95, 137].

Appendix B

Summary in Hungarian

Bevezetés

A 90-es években kezdett kibontakozni egy új kutatási irány az elosztott számítások területén, amelyet Grides számításoknak (Grid Computing [29]) neveztek el. A Grid rendszerek (számítóhálók) lényege a világ különböző tájain lévő számítási rendszerek virtuális egyesítése, nagyobb számítási kapacitás elérése érdekében. Az érdeklődés egyre nőtt ezen szakterület iránt: ezt bizonyítja a számos világméretű Grid kutatással foglalkozó projekt (pld. CoreGRID [95], EGEE [92], NextGRID [121], GEANT [97], KnowARC [104], EUAsiaGrid [94] és OSG [127]). Ekkor még a nagy számítási igényű feladatokkal rendelkező kutatók kétkedéssel tekintettek a Grideket hirdető, népszerűsítő fejlesztőkre, akik rövidebb futtatási időt és kényelmes kezelőfelületet ígértek. Mivel az elosztott számításokban alkalmazott korábbi technikák nem bizonyultak alkalmasnak a Grid rendszerek különféle kihívásainak megoldására, új kutatási irányok körvonalazódtak ki, melyek önálló kutatási területté emelték a Grides számításokat. A Grid rendszerek fejlődése során számos kutatási területről (pld. biológia, kémia, fizika) érkeztek felhasználók, akik a kezdeti nehézségek ellenére beléptek a Gridet alkalmazók körébe. A napjainkban látható statisztikák és kutatási eredmények azt mutatják, hogy helyesen cselekedtek. A mára elegendően stabil és megbízható Gridek kutatása a felhasználói igényekre összpontosít, hiszen ezen követelmények teljesítése elengedhetetlen a majdan üzleti célokat szolgáló Gridek számára.

A Grid rendszerek magját az ún. köztes réteg (Grid middleware [33]) adja, amelyet az egyes projektek elszigetelt módon kezdtek el kidolgozni. Az első, a gyakorlatban is elterjedt "de facto" szabványként kezelt köztes réteg a Globus Toolkit

[30] volt. A különböző projektek világszerte számos, a gyakorlatban működő ún. produkciós Grideket hoztak létre a közel tíz éves fejlesztések következtében (pld. HunGrid [110], NGS [120], EGEE [92], UNICORE [143], NorduGrid [122] és OSG [127]). Az így kialakult Gridek viszont eltérő megvalósítású köztes rétegekre épültek, mely a kutató és fejlesztő közösségek mellett a felhasználói csoportokat is elszigetelte. A napjainkban is elérhető Grid rendszerek népszerűsítésére több nemzetközi projekt speciális felhasználó-támogató csoportot [142, 98, 146, 103] hozott létre a tudományos alkalmazások gridesítésére. Az adaptált alkalmazások között előfordulnak olyan nagy méretű és komplexitású munkafolyamatok, amelyek lefuttatásához egyetlen Grid erőforrásai kevésnek bizonyulnak. Ezért a világhálóhoz hasonlóan, a jövőben egy együttműködő, világméretű Grid rendszer lesz csak képes kiszolgálni a növekvő méretű és igényű felhasználói közösségeket. Ehhez egyesíteni kell az elszeparált szigetekként működő Grideket, mely nagy kihívást jelent és új kutatási megközelítéseket kíván.

A Grid rendszereken belül az erőforrás-kezelő komponensek fejlesztésével foglalkozó kutatási területet érinti a leginkább a felhasználói igények felerősödése (pld. eltérő Grid erőforrások együttes használata, szerződések alkalmazása, stb.). Ez az értekezés az eltérő megvalósítású szolgáltatói Gridek együttműködésének (Grid Interoperability [70]) elérését tűzte ki célul a Grides erőforrás kezelés témakörében. A Grides együttműködés a különféle Grid infrastruktúrák áthidalását jelenti, amely lehetővé teszi, hogy egy adott Grid felhasználói képesek legyenek más Grid erőforrásait felhasználni alkalmazásaik futtatására és adataik megosztására a többi Grid felhasználóival. Bár napjainkra számos jól megtervezett, széles körben használt Grides erőforrás-kezelő rendszer (Resource Management System), Grid bróker [2] elérhető a felhasználói közösség számára, ezek az eszközök a Gridet megvalósító köztes réteg komponenseire, szolgáltatásaira épülnek, melyek kevéssé adnak lehetőséget az újonnan felmerült igények kielégítésére. Az elérhető Grides erőforrás-kezelő komponenseket más kutatócsoportok is vizsgálták [52], viszont ezen publikációk nem részletezik az együttműködés szempontjából fontos kapcsolatokat, felelősségi köröket és tulajdonságokat. A jelenlegi megvalósítások nagy része nem képes átlépni a köztes réteg alkalmazói korlátait, ezáltal csak a teljes Grid rendszer fejlesztésével azonos mértékben fejlődhetnek, mely igen lassú előrelépést és az új igények tekintetében radikális változtatásokat jelentenek. Emellett napjaink szolgáltatói Gridjei viszonylag elkülönített felhasználói közösséggel és fejlesztői csoporttal rendelkeznek, mely szintén az együttműködés elősegítésének útjában áll.

Az együttműködő Gridek problémájával nagy tekintélyű szakértői csoportok is foglalkoznak. Az egyik ilyen, Európában irányadó Grides szakértői csoport a Next Generation Grids Expert Group, amely az Európai Bizottság égisze alatt működik. Az európai Gridek jövőjéről szóló harmadik közleményükben [61], a 2010-ig megvalósítandó és azon túlmutató célokat, kutatási irányokat jelölték ki. Ebben a dokumentumban a webes és Grides technológiák konvergenciáját állapították meg, és egyben kijelölték az utat a szolgáltatás-orientált tudásalapú komponensek, ún. SOKUk (Service Oriented Knowledge Utility) fejlesztése felé, amelyeknek együttműködő, megbízható és hibatűrő működést megvalósító, megfelelő tudásbázissal rendelkező szolgáltatásoknak kell lenniük. Mindezen szakértői útmutatásokat figyelembe véve ez az értekezés olyan magas szintű brókerező szolgáltatást javasol az együttműködési probléma megoldására, amely a lehető legtöbb felhasználói igényt képes kielégíteni, és nem igényli a köztes réteg komponenseinek újratervezését.

Új tudományos eredmények

Kutatásaim során első célom a Grid brókerezés szakirodalmának mélyreható vizsgálata volt. Ezidőtájt a közel 10 éves Grid rendszerek már számos erőforrás-kezelő megoldásokkal rendelkeztek, azonban ezek az eszközök különféle Grid megvalósításra épültek, más elnevezéssel rendelkeztek és eltérő felhasználói igényeket céloztak meg. Az első tézis előkészítéseként megvizsgáltam a napjainkban elérhető, nagyobb felhasználói közösségek által használt Grides erőforrás brókerek működését, felépítését és gyakorlati tulajdonságaikat. Részletesen tanulmányoztam a különböző erőforrás-kezelő komponensek külső kapcsolatait és belső felépítésüket, és azonosítottam az eltérő felelősségi köröket és megnevezéseket egy Grid erőforrás-kezelő anatómia meghatározásával. Az ASM (Abstract State Machine) Grid formális modellt [60] felhasználva formalizáltam az azonosított Grides brókerező feladatköröket és együttműködési szinteket, melyek lehetővé teszik a Grid brókerező megoldások elkülönítését. Ezek az eredmények a következő tézishez vezettek:

> **I. Tézis. Felállítottam egy gyakorlati tulajdonságokon alapuló kategória rendszert, melyet felhasználva létrehoztam egy általános Grid bróker taxonómiát. Kidolgoztam egy Grid erőforrás-kezelő anatómiát, amely alapján formalizáltam a Grid brókerező szinteket felhasználva az ASM Grid modellt [60].**

A Grid rendszerek kutatásában napjaink legnagyobb kihívását az együttműködés [70] megteremtése jelenti. A bróker taxonómia is rámutat a brókerező módszerek és komponensek különbözőségére, míg az anatómia felfedi az együttműködés szempontjából fontos hasonlóságokat és kijelöli az együttműködés megteremtésének lehetőségét egy magasabb absztrakciós szinten. A taxonómiában vizsgált brókerek közül néhány képes alacsony szintű együttműködésre több Grid erőforrásainak elérésével. Gyakorlati példákon keresztül bemutattam az ezen az elven működő multi-Grid brókerezést bróker-kiterjesztéssel és portál használatával. Egy magasabb szintű együttműködést lehetővé tevő brókerezéshez szükség van egy brókereket leíró nyelvre a brókerek együttes kezeléséhez. A bróker taxonómia kategóriáit felhasználó, magas szintű adat-modellre épülő nyelv kidolgozását foglalja magába a második tézis.

II. Tézis. Létrehoztam egy olyan új, XML-alapú bróker-leíró nyelvet, a BPDL-t (Broker Property Description Language), mely felhasználásával egy magas szintű brókerező szolgáltatás képes tetszőleges számú, a bróker taxonómiába sorolható Grid brókert egy rendszerben kezelni.

A brókerek együttes kezelését megvalósító, meta-szinten működő, magas szintű Grides erőforrás-kezelő megoldást meta-brókerezésnek neveztem el. A következő, harmadik tézis keretében azonosítottam egy általános meta-brókerező megoldás követelményrendszerét a működéshez szükséges komponensek definiálásával, és kidolgoztam ezen absztrakt rendszer olyan megvalósítását, amely nem igényli az alkalmazott brókerek és Grid rendszerek módosítását.

III. Tézis. Meghatároztam egy általános meta-brókerező szolgáltatás követelményrendszerét, mely alapján megterveztem a rendszer meta-bróker architektúráját. Ez egy új absztrakciós szint bevezetésével lehetővé teszi a Grid rendszerek együttműködését tetszőleges brókerek integrálásával. Az architektúra terv alapján megvalósítottam az új GMBS (Grid Meta-Broker Service) meta-bróker szolgáltatás komponenseit.

A megvalósítás komponensei elvégzik a menedzselt brókerek teljesítményének és Gridjeik terheltségének monitorozását, szabványos interfészen keresztül biztosítják

a felhasználói interakciót és elvégzik az automatikus bróker-választást. A meta-brókerező módszer publikálása után hasonló megközelítések jelentek meg a szakirodalomban. Az első tézisben definiált formális együttműködési szintek segítségével összehasonlítottam ezeket a megközelítéseket. Kutatásom végső állomását a meta-bróker kiértékelése jelentette. A széles körben elterjedt és használt GridSim Toolkit [12] Grides szimulációs környezetet használtam fel a kiértékelő rendszer kidolgozásához. A negyedik tézis a GridSim-et kiegészítő, a meta-brókerezés vizsgálatát lehetővé tevő szimulációs környezet kidolgozását és a meta-bróker valós adatokkal történő kiértékelését tartalmazza, melyhez a valós szuperszámítógép és Grid futási adatokat tartalmazó Parallel és Grid Workloads Archive nyilvánosan elérhető adattárak adatfájljait használtam fel [129, 107].

IV. Tézis. A GridSim [12] szimulációs környezetre építve megterveztem és megvalósítottam egy új, meta-brókerezés vizsgálatát lehetővé tevő szimulációs rendszert. Ezt felhasználva elvégeztem a GMBS meta-bróker szolgáltatás teljesítmény elemzését valós párhuzamos szuperszámítógép és Grides erőforrások terheltségi adatainak alapján. A vizsgálattal bizonyítottam a meta-bróker szolgáltatás hatékonyságát.

A különböző módon felparaméterezett szimulációs kísérletek mindegyikében hatékonyabbnak bizonyult a brókereket együttműködő módon alkalmazó meta-brókerező szolgáltatás a hagyományos, elszigetelt bróker használattal szemben. A mérési eredmények alapján a GMBS meta-bróker szolgáltatás képes több, mint 10-szeres gyorsulás elérésére a véletlenszerű brókerválasztással szemben.

Összefoglalás

Napjainkban a számos kutatói közösség által használt Grid rendszerek további fejlődésének útjában áll az együttműködés hiánya. A Gridekre adaptált alkalmazások között megjelentek olyan nagy méretű és komplexitású munkafolyamatok, amelyek lefuttatásához egy Grid erőforrásai kevésnek bizonyulnak. Ennek a problémának a megoldására olyan fejlett erőforrás-kezelő megoldásokat mutattam be ebben a disszertációban, amelyek képesek a különféle Grid rendszerek erőforrásait együttesen és automatizált módon felhasználni a komplex felhasználói igények kielégítésére.

A P-GRADE portal [46] és a gUSE/WS-PGRADE rendszer [43] brókerezéssel kapcsolatos szolgáltatásai az értekezésben kidolgozott módszereken alapulnak. En-

nek megfelelően a disszertáció eredményei a következő Európai Uniós projektekben hasznosulnak: SHIWA projekt [138], EDGI projekt [91], CancerGrid projekt [102] és a GASuC projekt [103]. A következő országos projektek szintén használják a portálokat: UK ProSim projekt [134], MoSGrid projekt [118], valamint az ETH Zurich egy biológus projektje [141].

A tézisek tudományos eredményeit számos nemzetközi folyóiratban, konferencia és workshop cikkben publikáltam, és különféle tudományos fórumokon adtam elő. A disszertáció publikációi több későbbi kutatás alapjául szolgáltak, amelyet a sok független hivatkozás fémjelez világszerte. Az értekezésben bemutatott kutatási eredmények nagy része a CoreGRID és S-CUBE európai kiválósági hálózatokban (Network of Excellence) [95, 137] történő aktív részvétel sikere.

Appendix C

Additional information

In Section 4.2 of Chapter 4 I introduced an extendable Broker Property Description Language (BPDL) to express metadata about brokers. After revising the schema of this language description I created BPDL 2.0, and the Meta-Broker Scheduling Description Language (MBSDL) that is able to express special attributes of the different job description documents and can be used as an extension of JSDL. The XML schemas of these documents are presented next.

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:bpdl="uri:BrokerPropertyDescriptionLanguage"
       xmlns:mbsdl="uri:MBSchedulingDescriptionLanguage"
       targetNamespace="uri:BrokerPropertyDescriptionLanguage"
       elementFormDefault="qualified" attributeFormDefault="unqualified">
4    <xsd:import namespace="uri:MBSchedulingDescriptionLanguage"
         schemaLocation="mbsdl.xsd"/>
5    <xsd:element name="BPDL" type="bpdl:BPDL_Type">
6      <xsd:annotation>
7        <xsd:documentation>Broker Property Description Language
             2.0</xsd:documentation>
8      </xsd:annotation>
9    </xsd:element>
10   <xsd:complexType name="BPDL_Type">
11     <xsd:sequence>
12       <xsd:any namespace="##other" processContents="lax"/>
13       <xsd:element name="BrokerID" type="bpdl:BrokerID_Type"/>
14       <xsd:element name="Interface" type="bpdl:InterfaceType"
             maxOccurs="unbounded"/>
```

```
15        <xsd:element name="Monitoring" type="bpdl:Monitoring_Type"/>
16        <xsd:element name="Security" type="bpdl:Security_Type"/>
17        <xsd:element name="PerformanceMetrics"
              type="bpdl:PerformanceMetrics_Type"/>
18        <xsd:element ref="mbsdl:SDL"/>
19      </xsd:sequence>
20      <xsd:attribute name="name" type="xsd:NCName" use="required"/>
21      <xsd:attribute name="description" type="xsd:string"
            use="optional"/>
22      <xsd:attribute name="targetNameSpace" type="xsd:anyURI"
            use="optional"/>
23      <xsd:anyAttribute namespace="##other" processContents="lax"/>
24    </xsd:complexType>
25    <xsd:complexType name="BrokerID_Type">
26      <xsd:simpleContent>
27        <xsd:extension base="xsd:string">
28          <xsd:anyAttribute namespace="##other" processContents="lax"/>
29        </xsd:extension>
30      </xsd:simpleContent>
31    </xsd:complexType>
32    <xsd:complexType name="InterfaceType">
33      <xsd:sequence>
34        <xsd:element name="type" type="bpdl:InterfacesEnumeration"/>
35        <xsd:element name="name" type="xsd:string"/>
36        <xsd:element name="description" type="xsd:string"
              minOccurs="0"/>
37        <xsd:element name="Parameters" minOccurs="0">
38          <xsd:complexType>
39            <xsd:sequence>
40              <xsd:element name="Parameter" maxOccurs="unbounded">
41                <xsd:complexType>
42                  <xsd:sequence>
43                    <xsd:element name="description" minOccurs="0"/>
44                  </xsd:sequence>
45                  <xsd:attribute name="name" type="xsd:NCName"/>
46                  <xsd:attribute name="type" type="xsd:NCName"/>
47                </xsd:complexType>
48              </xsd:element>
49            </xsd:sequence>
50          </xsd:complexType>
51        </xsd:element>
52        <xsd:element name="ReturnedValue" minOccurs="0">
```

```
53          <xsd:complexType>
54            <xsd:sequence>
55              <xsd:element name="description" minOccurs="0"/>
56            </xsd:sequence>
57            <xsd:attribute name="name" type="xsd:NCName"/>
58            <xsd:attribute name="type" type="xsd:NCName"/>
59          </xsd:complexType>
60        </xsd:element>
61      </xsd:sequence>
62    </xsd:complexType>
63    <xsd:complexType name="MonitoringMetric_Type">
64      <xsd:sequence>
65        <xsd:element name="Name" type="xsd:string"/>
66        <xsd:element name="Description" type="xsd:string"/>
67      </xsd:sequence>
68      <xsd:attribute name="MetricType"
          type="bpdl:MonitoringInfoEnumeration" use="required"/>
69      <xsd:anyAttribute namespace="##other"/>
70    </xsd:complexType>
71    <xsd:complexType name="Monitoring_Type">
72      <xsd:sequence>
73        <xsd:element name="Metric" type="bpdl:MonitoringMetric_Type"
            maxOccurs="unbounded"/>
74      </xsd:sequence>
75      <xsd:attribute name="accessMethod" type="xsd:string"
          use="optional"/>
76      <xsd:anyAttribute namespace="##other"/>
77    </xsd:complexType>
78    <xsd:complexType name="Security_Type">
79      <xsd:choice>
80        <xsd:element name="MyProxy" type="bpdl:MyProxy_Type"/>
81        <xsd:element name="OtherSecurity"
            type="bpdl:OtherSecurity_Type"/>
82      </xsd:choice>
83      <xsd:anyAttribute namespace="##other"/>
84    </xsd:complexType>
85    <xsd:complexType name="MyProxy_Type">
86      <xsd:sequence>
87        <xsd:element name="IsSupported" type="xsd:boolean"/>
88        <xsd:element name="ServerName" type="xsd:string"
            minOccurs="0"/>
89        <xsd:element name="PortNumber" type="xsd:int" minOccurs="0"/>
```

```
90      </xsd:sequence>
91      <xsd:anyAttribute namespace="##other"/>
92    </xsd:complexType>
93    <xsd:complexType name="OtherSecurity_Type">
94      <xsd:sequence>
95        <xsd:element name="Details" type="xsd:string"/>
96      </xsd:sequence>
97      <xsd:attribute name="name" type="xsd:NCName"/>
98      <xsd:anyAttribute namespace="##other"/>
99    </xsd:complexType>
100   <xsd:complexType name="PerformanceMetrics_Type">
101     <xsd:sequence>
102       <xsd:element name="AVGWaitingTime"
              type="bpdl:PerformanceMetric_Type"/>
103       <xsd:element name="AVGSlowdown"
              type="bpdl:PerformanceMetric_Type"/>
104       <xsd:element name="FinishedJobs"
              type="bpdl:PerformanceMetric_Type"/>
105       <xsd:element name="FailedJobs"
              type="bpdl:PerformanceMetric_Type"/>
106       <xsd:element name="OtherMetric"
              type="bpdl:PerformanceMetric_Type" maxOccurs="unbounded"/>
107       <xsd:element name="Prediction"
              type="bpdl:PerformanceMetric_Type" minOccurs="0"
              maxOccurs="unbounded"/>
108     </xsd:sequence>
109     <xsd:anyAttribute namespace="##other"/>
110   </xsd:complexType>
111   <xsd:complexType name="PerformanceMetric_Type">
112     <xsd:sequence>
113       <xsd:element name="name" type="xsd:string"/>
114       <xsd:element name="description" type="xsd:string"/>
115       <xsd:element name="value" type="xsd:string"/>
116     </xsd:sequence>
117     <xsd:anyAttribute namespace="##other"/>
118   </xsd:complexType>
119   <xsd:simpleType name="InterfacesEnumeration">
120     <xsd:restriction base="xsd:string">
121       <xsd:enumeration value="Submit"/>
122       <xsd:enumeration value="Cancel"/>
123       <xsd:enumeration value="Suspend"/>
124       <xsd:enumeration value="Resume"/>
```

```
125         <xsd:enumeration value="Migrate"/>
126         <xsd:enumeration value="other"/>
127       </xsd:restriction>
128     </xsd:simpleType>
129     <xsd:simpleType name="MonitoringInfoEnumeration">
130       <xsd:restriction base="xsd:string">
131         <xsd:enumeration value="StaticInfo"/>
132         <xsd:enumeration value="DynamicInfo"/>
133         <xsd:enumeration value="AggregatedInfo"/>
134         <xsd:enumeration value="other"/>
135       </xsd:restriction>
136     </xsd:simpleType>
137 </xsd:schema>
138
139
140 <?xml version="1.0" encoding="UTF-8"?>
141 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:mbsdl="uri:MBSchedulingDescriptionLanguage"
      targetNamespace="uri:MBSchedulingDescriptionLanguage"
      elementFormDefault="qualified" attributeFormDefault="unqualified">
142     <xsd:element name="SDL" type="mbsdl:SDL_Type">
143       <xsd:annotation>
144         <xsd:documentation>MB Scheduling Description
              Language</xsd:documentation>
145       </xsd:annotation>
146     </xsd:element>
147     <xsd:complexType name="SDL_Type">
148       <xsd:sequence>
149         <xsd:any namespace="##other" processContents="lax"/>
150         <xsd:element name="Constraints" type="mbsdl:Constraints_Type"/>
151         <xsd:element name="QoS" type="mbsdl:QoS_Type"/>
152         <xsd:element name="Policy" type="mbsdl:Policy_Type"/>
153       </xsd:sequence>
154       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
155       <xsd:attribute name="description" type="xsd:string"
            use="optional"/>
156       <xsd:attribute name="targetNameSpace" type="xsd:anyURI"
            use="optional"/>
157       <xsd:anyAttribute namespace="##other" processContents="lax"/>
158     </xsd:complexType>
159     <xsd:complexType name="Constraints_Type">
160       <xsd:sequence>
```

```
161         <xsd:any namespace="##other"/>
162         <xsd:element name="Middleware" type="mbsdl:Middleware_Type"
                maxOccurs="unbounded"/>
163         <xsd:element name="JobType" type="mbsdl:JobTypeEnumeration"
                maxOccurs="unbounded"/>
164         <xsd:element name="Time" type="mbsdl:Time_Type"/>
165         <xsd:element name="Budget" type="xsd:long"/>
166         <xsd:element name="RemoteFileAccess"
                type="mbsdl:RemoteFileAccessEnumeration" minOccurs="0"
                maxOccurs="unbounded"/>
167         <xsd:element name="OtherConstraint" type="mbsdl:Other_Type"
                maxOccurs="unbounded"/>
168       </xsd:sequence>
169     </xsd:complexType>
170     <xsd:complexType name="Middleware_Type">
171       <xsd:sequence>
172         <xsd:element name="GridName" type="mbsdl:GridNameEnumeration"
                minOccurs="0"/>
173         <xsd:element name="ProxyName" type="xsd:string" minOccurs="0"/>
174         <xsd:element name="MYProxy" type="mbsdl:MyProxy_Type"
                minOccurs="0"/>
175         <xsd:element name="VirtualOrganisation"
                type="mbsdl:VirtualOrganisation_Type" minOccurs="0"
                maxOccurs="unbounded"/>
176         <xsd:element name="InformationSystem"
                type="mbsdl:InformationSystem_Type" minOccurs="0"/>
177         <xsd:any namespace="##other" minOccurs="0"/>
178       </xsd:sequence>
179       <xsd:anyAttribute namespace="##other" processContents="lax"/>
180     </xsd:complexType>
181     <xsd:complexType name="VirtualOrganisation_Type">
182       <xsd:sequence>
183         <xsd:element name="InformationSystem"
                type="mbsdl:InformationSystem_Type"/>
184         <xsd:element name="ProxyName" type="xsd:string" minOccurs="0"/>
185         <xsd:any namespace="##other" minOccurs="0"/>
186       </xsd:sequence>
187       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
188       <xsd:anyAttribute namespace="##other" processContents="lax"/>
189     </xsd:complexType>
190     <xsd:complexType name="InformationSystem_Type">
191       <xsd:sequence>
```

```
192         <xsd:element name="MDS" type="xsd:string" minOccurs="0"/>
193         <xsd:element name="BDII" type="xsd:string" minOccurs="0"/>
194         <xsd:element name="WebMDS" type="xsd:string" minOccurs="0"/>
195         <xsd:any namespace="##other" minOccurs="0"/>
196       </xsd:sequence>
197       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
198       <xsd:anyAttribute namespace="##other" processContents="lax"/>
199     </xsd:complexType>
200     <xsd:complexType name="QoS_Type">
201       <xsd:sequence>
202         <xsd:any namespace="##other"/>
203         <xsd:element name="Agreement" type="mbsdl:Agreement_Type"
              minOccurs="0" maxOccurs="unbounded"/>
204         <xsd:element name="FaultTorelanceMechanisms"
              type="mbsdl:FaultToleranceEnumeration"
              maxOccurs="unbounded"/>
205         <xsd:element name="AdvanceReservation"
              type="mbsdl:AdvanceReservation_Type" minOccurs="0"/>
206         <xsd:element name="Priority" type="xsd:string" minOccurs="0"/>
207         <xsd:element name="GridAccessControl" type="xsd:string"
              minOccurs="0"/>
208         <xsd:element name="EmailNotification" type="xsd:string"
              minOccurs="0"/>
209       </xsd:sequence>
210       <xsd:anyAttribute namespace="##other" processContents="lax"/>
211     </xsd:complexType>
212     <xsd:complexType name="BrokerName_Type">
213       <xsd:simpleContent>
214         <xsd:extension base="xsd:string">
215           <xsd:anyAttribute namespace="##other" processContents="lax"/>
216         </xsd:extension>
217       </xsd:simpleContent>
218     </xsd:complexType>
219     <xsd:complexType name="Policy_Type">
220       <xsd:sequence>
221         <xsd:element name="PolicyName" type="mbsdl:PolicyEnumeration"
              minOccurs="0"/>
222         <xsd:element name="OtherPolicy" type="mbsdl:Other_Type"
              minOccurs="0"/>
223         <xsd:element name="LRMSPolicy" type="mbsdl:Other_Type"
              minOccurs="0"/>
224       </xsd:sequence>
```

```
225     <xsd:anyAttribute namespace="##other"/>
226   </xsd:complexType>
227   <xsd:complexType name="Time_Type">
228     <xsd:sequence>
229       <xsd:element name="StartTime" type="xsd:date"/>
230       <xsd:element name="Duration" type="xsd:long"/>
231       <xsd:element name="TimeOut" type="xsd:long"/>
232     </xsd:sequence>
233     <xsd:anyAttribute namespace="##other"/>
234   </xsd:complexType>
235   <xsd:complexType name="Other_Type">
236     <xsd:sequence>
237       <xsd:element name="Name" type="xsd:string"/>
238       <xsd:element name="Value" type="xsd:string"/>
239     </xsd:sequence>
240     <xsd:anyAttribute namespace="##other"/>
241   </xsd:complexType>
242   <xsd:complexType name="MyProxy_Type">
243     <xsd:sequence>
244       <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
245       <xsd:element name="ServerName" type="xsd:string"/>
246       <xsd:element name="PortNumber" type="xsd:int" minOccurs="0"/>
247     </xsd:sequence>
248     <xsd:anyAttribute namespace="##other"/>
249   </xsd:complexType>
250   <xsd:complexType name="Agreement_Type">
251     <xsd:sequence>
252       <xsd:element name="Target" type="xsd:anyURI"/>
253       <xsd:element name="ConfidenceLevel"
              type="xsd:positiveInteger"/>
254     </xsd:sequence>
255     <xsd:anyAttribute namespace="##other"/>
256   </xsd:complexType>
257   <xsd:complexType name="AdvanceReservation_Type">
258     <xsd:sequence>
259       <xsd:element name="ResourceName" type="xsd:string"/>
260       <xsd:element name="Date" type="xsd:date"/>
261     </xsd:sequence>
262     <xsd:anyAttribute namespace="##other"/>
263   </xsd:complexType>
264   <xsd:simpleType name="GridNameEnumeration">
265     <xsd:restriction base="xsd:string">
```

```
266         <xsd:enumeration value="GT2"/>
267         <xsd:enumeration value="GT3"/>
268         <xsd:enumeration value="GT4"/>
269         <xsd:enumeration value="EGEE-LCG-2"/>
270         <xsd:enumeration value="EGEE-gLite"/>
271         <xsd:enumeration value="NorduGrid"/>
272         <xsd:enumeration value="Unicore"/>
273       </xsd:restriction>
274     </xsd:simpleType>
275     <xsd:simpleType name="JobTypeEnumeration">
276       <xsd:restriction base="xsd:string">
277         <xsd:enumeration value="Serial"/>
278         <xsd:enumeration value="Mpi"/>
279         <xsd:enumeration value="Pvm"/>
280         <xsd:enumeration value="Checkpointable"/>
281         <xsd:enumeration value="Interactive"/>
282         <xsd:enumeration value="Threads"/>
283         <xsd:enumeration value="OpenMP"/>
284         <xsd:enumeration value="Mpi+OpenMP"/>
285         <xsd:enumeration value="Caf"/>
286         <xsd:enumeration value="Upc"/>
287       </xsd:restriction>
288     </xsd:simpleType>
289     <xsd:simpleType name="RemoteFileAccessEnumeration">
290       <xsd:restriction base="xsd:string">
291         <xsd:enumeration value="GridFTP"/>
292         <xsd:enumeration value="RFT"/>
293         <xsd:enumeration value="GASS"/>
294         <xsd:enumeration value="Unicore"/>
295         <xsd:enumeration value="SRB"/>
296         <xsd:enumeration value="EGEE-LFN"/>
297       </xsd:restriction>
298     </xsd:simpleType>
299     <xsd:simpleType name="PolicyEnumeration">
300       <xsd:restriction base="xsd:string">
301         <xsd:enumeration value="ScheduleByCpu"/>
302         <xsd:enumeration value="ScheduleByMemory"/>
303         <xsd:enumeration value="ScheduleByDiskSize"/>
304         <xsd:enumeration value="RandomHost"/>
305       </xsd:restriction>
306     </xsd:simpleType>
307     <xsd:simpleType name="FaultToleranceEnumeration">
```

```
308       <xsd:restriction base="xsd:string">
309         <xsd:enumeration value="Checkpointing"/>
310         <xsd:enumeration value="Rescheduling"/>
311         <xsd:enumeration value="Replication"/>
312       </xsd:restriction>
313     </xsd:simpleType>
314 </xsd:schema>
```

In Section 4.2 of Chapter 4 I also discussed the components of a proposed meta-brokering service called GMBS. Figure C.1 is intended to give an overview of the implementation of GMBS through a UML class diagram representing the main components of the system.

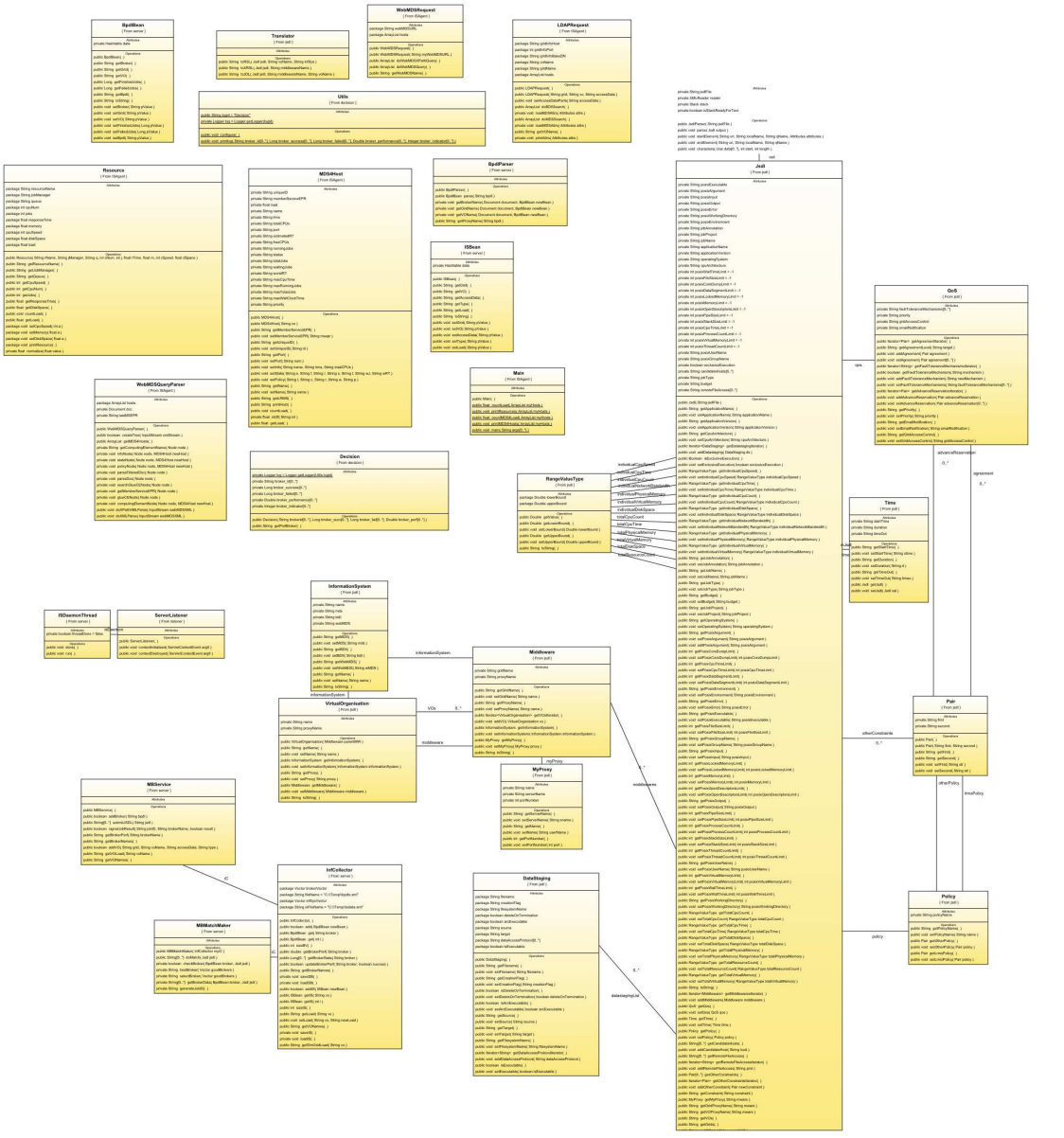

Figure C.1: UML class diagram of GMBS.

Bibliography

[1] M. Addis, et al: "Experiences with eScience workflow specification and enactment in bioinformatics", in Proc. of UK e-Science All Hands Meeting (Editor: Simon J. Cox), 2003.

[2] E. Afgan, "Role of the Resource Broker in the Grid", Proceedings of the 42nd annual Southeast regional conference, 2004.

[3] M. Altenhofen, A. Friesen and J. Lemcke, "ASMs in Service Oriented Architectures", Journal of Universal Computer Science, vol. 14, no. 12, pp. 2034-2058, 2008.

[4] M. D. Assuncao, R. Buyya and S. Venugopal, "InterGrid: A Case for Internetworking Islands of Grids", Concurrency and Computation: Practice and Experience (CCPE), Online ISSN: 1532-0634; Print ISSN: 1532-0626, Wiley Press, New York, USA, Jul. 16 2007.

[5] W. H. Bell, D. G. Cameron, R. Carvajal-Schiaffino, A. P. Millar, K. Stockinger and F. Zini, "Evaluation of an Economy-Based File Replication Strategy for a Data Grid", In International Workshop on Agent based Cluster and Grid Computing at CCGrid 2003, IEEE Computer Society Press, May 2003.

[6] Börger, E., and R. Stark, "Abstract State Machines. A method for High-level System Design and Analysis", Springer, 2003.

[7] Börger, E. and B. Thalheim, "Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach", In Proceedings of the 1st international Conference on Abstract State Machines, B and Z, Lecture Notes In Computer Science, vol. 5238, Springer-Verlag, pp. 24-38, 2008.

[8] Bratosin, C., W. Aalst, N. Sidorova, and N. Trcka, A Reference Model for Grid Architectures and Its Analysis, In Proceedings of the OTM 2008 Confederated international Conferences, Lecture Notes In Computer Science, vol. 5331. Springer-Verlag, pp. 898-913, 2008.

[9] J. Brooke, D. Fellows, K. Garwood, C. Goble, "Semantic matching of Grid resource descriptions", UoM. 2nd European Across-Grids Conference (AxGrids 2004), 2004.

[10] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch, "A National-Scale Authentication Infrastructure", IEEE Computer, 33 (12), pp. 60-66, 2000.

[11] R. Buyya, D. Abramson, J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), IEEE Computer Society Press, 2000.

[12] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", Concurrency and Computation: Practice and Experience., pp. 1175-1220, Volume 14, Issue 13-15, 2002.

[13] R. Buyya, S. Venugopal, R. Ranjan, and C. S. Yeo, "The Gridbus Middleware for Market-Oriented Computing", Market Oriented Grid and Utility Computing, Wiley Press, Hoboken, New Jersey, USA, Oct. 2009.

[14] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic. "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility", Future Generation Computer Systems, ISSN: 0167-739X, Elsevier Science, Amsterdam, The Netherlands, 2009.

[15] H. Casanova, G. Obertelli, F. Berman, R. Wolski, "The AppLeS parameter sweep template: user-level middleware for the grid", In Proceedings of the ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2000.

[16] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, "GridFlow: WorkFlow Management for Grid Computing", in Proc. of the 3rd IEEE/ACM International

Symposium on Cluster Computing and the Grid (CCGRID'03), pp. 198-205, 2003.

[17] E. Deelman, et al, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, Vol.1, no. 1, pp. 25-39, 2003.

[18] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter, P. Kacsuk, "GEMLCA: Running Legacy Code Applications as Grid Services", Journal of Grid Computing, Vol. 3 No. 1-2, pp. 75-90, 2005.

[19] J. D. Dombi, A. Kertész, "Scheduling solution for Grid Meta-brokering using Pliant system", In Proc. of. 2nd International Conference on Agents and Artificial Intelligence (ICAART '10), pp. 46-53, Valencia, Spain, 22-24 January, 2010.

[20] J. D. Dombi, A. Kertész, "Advanced Scheduling Techniques with the Pliant System for High-Level Grid Brokering", Communications in Computer and Information Science (CCIS), Vol. 129, Springer-Verlag Berlin Heidelberg, pp. 173–185, 2011.

[21] C. Dumitrescu, I. Foster, "GRUBER: A Grid Resource Usage SLA Broker", 11th International Euro-Par Conference, LNCS 3648, pp. 465-474, 2005.

[22] C. L. Dumitrescu and I. Foster, "Gangsim: A simulator for grid scheduling studies", In proc. of IEEE International Symposium on Cluster Computing and Grid, pp 1151-1158, 2005.

[23] E. Elmroth and J. Tordsson, "An Interoperable Standards-based Grid Resource Broker and Job Submission Service", First IEEE Conference on e-Science and Grid Computing, IEEE Computer Society Press, pp. 212-220, 2005.

[24] E. Elmroth and J. Tordsson, "A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability", Concurrency and Computation: Practice and Experience, Vol. 25, No. 18, pp. 2298-2335, 2009.

[25] D. W. Erwin and D. F. Snelling., "UNICORE: A Grid Computing Environment", In Lecture Notes in Computer Science, volume 2150, Springer, pp. 825-834, 2001.

[26] Y. Etsion, D. Tsafrir, "A short survey of commercial cluster batch schedulers", Technical Report 2005-13, School of Computer Science and Engineering, the Hebrew University, May 2005, Jerusalem, Israel, 2005.

[27] Z. Farkas, P. Kacsuk, G. Gombás and Z. Balaton, "Generic MPI program execution support at job and workflow level by the P-GRADE Grid portal", Journal of Grid Computing, submitted in 2006.

[28] L. Field, "Getting Grids to work together", CERN Computer Newsletter, 41 (5), pp. 8-9, 2006.

[29] I. Foster, C. Kesselman, "Computational Grids, The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann, pp. 15-52, 1998.

[30] I. Foster, C. Kesselman, "The Globus project: A status report", in Proc. of the Heterogeneous Computing Workshop, IEEE Computer Society Press, pp. 4-18, 1998.

[31] I. Foster and C. Kesselman, "The Grid 2: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers Inc., 2003.

[32] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A security architecture for computational grids", In Proceedings of the 5th ACM Conference on Computer and Communications Security, ACM, pp. 83-92, 1998.

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

[34] E. Frizziero, M. Gulmini, F. Lelli, G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato, and S.Traldi, "Instrument Element: A New Grid component that Enables the Control of Remote Instrumentation". In Proceedings of the Sixth IEEE international Symposium on Cluster Computing and the Grid (CCGRID'06), Volume 00, May 16-19, IEEE Computer Society, Washington, DC, 52, 2006.

[35] M. R. Garey and D. S. Johnson, "Computers and Intractability; a Guide to the Theory of Np-Completeness", W. H. Freeman & Co., New York, USA, 1979.

[36] F. Guim, J. Corbalan, J. Labarta, "Modeling the impact of resource sharing in Backfilling Policies using the Alvio Simulator", 15th Annual Meeting of the IEEE

International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007.

[37] F. Howell and R. McNab, "SimJava: A discrete event simulation library for Java", In Proc. of the International Conference on Web-Based Modeling and Simulation, San Diego, USA, 1998.

[38] E. Huedo, R. S. Montero, I. M. Llorente, "A framework for adaptive execution in grids", Software: Practice and Experience. vol. 34, 7, pp. 631-651, 2004.

[39] Institute of Electrical and Electronics Engineers, "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries", New York, 1990.

[40] A. Iosup, T. Tannenbaum, M. Farrellee, D. Epema, and M. Livny, Inter-operating grids through Delegated MatchMaking. Sci. Program. 16, 2-3, pp. 233-253, 2008.

[41] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, D. H.J. Epema, "The Grid Workloads Archive", Future Generation Computer Systems, Volume 24, Issue 7, pages 672-686, July 2008.

[42] P. Kacsuk, Z. Farkas and G. Fedak, "Towards making BOINC and EGEE interoperable", In Proc. of the International Grid Interoperability and Interoperation Workshop (IGIIW), Indianapolis, 2008.

[43] P. Kacsuk, K. Karóczkai, G. Hermann, G. Sipos, and J. Kovács, "WS-PGRADE: Supporting parameter sweep applications in workflows", Proc. of the 3rd Workshop on Workflows in Support of Large-Scale Science (in conjunction with SC08), Austin, 2008.

[44] P. Kacsuk and T. Kiss, "Towards a scientific workflow-oriented computational World Wide Grid", Technical report, TR-115, CoreGRID – Network of Excellence, December 2007.

[45] P. Kacsuk, T. Kiss, G. Sipos, "Solving the grid interoperability problem by P-GRADE portal at workflow level", Future Generation Computer Systems, Volume 24, Issue 7, Pages 744-751, 2008.

[46] P. Kacsuk, G. Sipos, "Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal", Journal of Grid Computing, Volume 3, num. 3-4, pp. 221-238, 2006.

[47] A. Kertész, F. Ötvös, P. Kacsuk, "Gridifying the TINKER conformer generator application for gLite Grid", In proc. of 1st Workshop on High Performance Bioinformatics and Biomedicine (HiBB'10) in conjunction with Euro-Par 2010, Ischia, Italy, August 31 - September 3, 2010.

[48] A. Kertész, Z. Farkas, P. Kacsuk, "Multi-level Brokering Solution for Interoperating Service and Desktop Grids", In proc. of CoreGRID/ERCIM Workshop on Grids, Clouds and P2P Computing in conjunction with Euro-Par 2010, Ischia, Italy, August 31 - September 3, 2010.

[49] A. Kertész, P. Kacsuk, A. Iosup and D. H.J. Epema, "Investigating peer-to-peer meta-brokering in Grids", Technical report, TR-0170, Institute on Resource Management and Scheduling, CoreGRID – Network of Excellence, August 2008.

[50] A. Kertész, G. Kecskeméti, I. Brandic, "Autonomic SLA-aware Service Virtualization for Distributed Systems", In proceedings of the 19th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, Ayia Napa, Cyprus, February, 2011.

[51] Y. Kim, J. Yu, J. Hahm, J. Kim, et al., "Design and Implementation of an OGSI-Compliant Grid Broker Service", Proc. of CCGrid, 2004.

[52] K. Krauter, R. Buyya, M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", Softw., Pract. Exper., vol. 32, pp. 135-164, 2002.

[53] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, J. Pukacki, Dynamic grid scheduling with job migration and rescheduling in the GridLab resource management system, Scientific Programming. IOS Press, Volume 12, Number 4, pp. 263-273, 2004.

[54] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, "Grid scheduling simulations with GSSIM", In proc. of 13th International Conference on Parallel and Distributed Systems (ICPADS'07), pp. 1-8, 2007.

[55] K. Leal, E. Huedo, I. M. Llorente, "A decentralized model for scheduling independent tasks in Federated Grids", Future Generation Computer Systems, Volume 25, Issue 8, pp. 840-852, 2009.

[56] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?", Springer LNCS, Volume 3277, pp. 253-263, 2005.

[57] A. Legrand, L. Marchal, H. Casanova, "Scheduling distributed applications: The SimGrid simulation framework", In Proceedings 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003), Tokyo, Japan, 2003.

[58] L. Cs. Lőrincz, A. Ulbert, Z. Horváth, T. Kozsik, "Towards an Agent Integrated Speculative Scheduling Service", In Distributed and Parallel Systems, Springer US, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'06), pp. 211-222, 2007.

[59] H. Mohamed and D. Epema, KOALA: a co-allocating grid scheduler, Concurrency and Computation: Practice and Experience, Vol. 22, No. 16, pp. 1851-1876, 2008.

[60] Zs. Németh, and V. Sunderam, Characterizing Grids: Attributes, Definitions, and Formalisms, Journal of Grid Computing, vol. 1, pp. 9-23, 2003.

[61] Next Generation Grids Expert Group Report no. 3, "Future for European Grids: GRIDs and Service Oriented Knowledge Utilities – Vision and Research Directions 2010 and Beyond", NGG3, December 2006.

[62] F. Neubauer, A. Hoheisel and J. Geiler, "Workflow-based Grid applications", Future Generation Computer Systems, pp. 6-15, Volume 22, Issues 1-2, January 2006.

[63] J, Novotny, M. Russell, O. Wehrens: "Grid-Sphere: A Portal Framework for Building Collaborations" in Proc. of the 1st International Workshop on Middleware in Grid Computing, Rio de Janeiro, Brazil, 2003.

[64] J. Novotny, S. Tuecke, V. Welch, "An Online Credential Repository for the Grid: MyProxy", in Proc. of 10th IEEE International. Symposium on High Performance Distributed Computing, 2001.

[65] M. Parkin, R. M. Badia and J. Martrat, "A Comparison of SLA Use in Six of the European Commissions FP6 Project", CoreGRID Technical report no. TR-0129, Institute on Resource Management and Scheduling, CoreGRID – Network of Excellence, 2008.

[66] D. Pasztuhov, I. Szeberényi, "The new architecture of CONFLET system", NIIF, Networkshop 2007. Online: https://nws.niif.hu/ncd2007/docs/aen/036.pdf.

[67] A. Pugliese, D. Talia and R. Yahyapour, "Modeling and Supporting Grid Scheduling", CoreGrid Technical Report no. 56, August 2006.

[68] M. Rambadt, Ph. Weider: UNICORE – Globus: Interoperability of Grid Infrastructures, Proceedings of Cray User Group Summit 2002, Manchester, 2002.

[69] J. M. Ramirez-Alcaraz, A. Tchernykh, R. Yahyapour, U. Schwiegelshohn, A. Quezada-Pina, J. L. Gonzalez-Garcia, A. Hirales-Carbajal, "User Run Time Estimate Unaware Online Job Scheduling in Hierarchical Grids", Submitted to Journal of Grid Computing, 2010.

[70] M. Riedel et al., "Interoperation of World-Wide Production e-Science Infrastructures", Concurrency and Computation: Practice and Experience, Volume 21, Issue 8, pp. 961-990, 2009.

[71] I. Rodero, J. Corbalan, R.M. Badia, J. Labarta, "eNANOS Grid Resource Broker", P.M.A. Sloot et al. (Eds.): EGC 2005, LNCS 3470, pp. 111-121, ISBN: 3-540-26918-5, Amsterdam, The Netherlands, 14-16 February, 2005.

[72] I. Rodero, F. Guim, J. Corbalan, L.L. Fong, Y.G. Liu, S.M. Sadjadi, "Looking for an Evolution of Grid Scheduling: Meta-brokering", Coregrid Workshop in Grid Middleware'07, Dresden, Germany, June 2007.

[73] P. Saiz, L. Aphecetche, P. Buncic, R. Piskac, J. E. Revsbech, V. Sego, AliEn–ALICE environment on the GRID, Nuclear Instruments and Methods in Physics Research, Volume 502, Issues 2-3, pp. 437-440, 2003.

[74] U. Schwiegelshohn, A. Tchernykh, R. Yahyapour, "Online scheduling in grids", 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), pp. 1-10, 2008.

[75] J. Seidel, O. Waldrich, W. Ziegler, P. Wieder and R. Yahyapour, "Using SLA for resource management and scheduling – a survey", Technical report, TR-0096, Institute on Resource Management and Scheduling, CoreGRID – Network of Excellence, August 2007.

[76] G. Singh et al, "The Pegasus Portal: Web Based Grid Computing", in Proc. of 20th Annual ACM Symposium on Applied Computing, Santa Fe, New Mexico, 2005.

[77] O. Smirnova et al., "The NorduGrid Architecture And Middleware for Scientific Applications", Springer-Verlag, LNCS 2657, 2003.

[78] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, J. Dongarra, "MPI: The Complete Reference", MIT Press, 1995.

[79] B. Sundaram and B. M. Chapman, "XML-Based Policy Engine Framework for Usage Policy Management in Grids", In Proceedings of the Third international Workshop on Grid Computing, LNCS vol. 2536, Springer-Verlag, pp. 194-198, 2002.

[80] V. Sunderam, J. Dongarra, "PVM: A framework for parallel distributed computing", Concurrency: Practice and Experience, 2(4), pp. 315-339, 1990.

[81] I. Taylor, et al., "Grid Enabling Applications Using Triana", Workshop on Grid Applications and Programming Tools, Seattle, 2003.

[82] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience", Concurrency and Computation: Practice and Experience, pp. 323-356, Volume 17, Issue 2-4, 2005.

[83] J.D. Ullman, "NP-complete scheduling problems", Journal of Computer and System Sciences, Volume 10, Issue 3, pp. 384-393, 1975.

[84] E. Urbah, P. Kacsuk, Z. Farkas, G. Fedak, G. Kecskemeti, O. Lodygensky, A. Marosi, Z. Balaton, G. Caillat, G. Gombás, Á. Kornafeld, J. Kovács, H. He, R. Lovas, "EDGeS: Bridging EGEE to BOINC and XtremWeb", In Journal of Grid Computing, Special Issue: Grid Interoperability, vol 7, issue 3, pp. 335-354, 2009.

[85] C. Vazquez, E. Huedo, R. S. Montero et I. M. Llorente, "Federation of TeraGrid, EGEE and OSG Infrastructures through a Metascheduler", Future Generation Computer Systems 26, pp. 979-985, 2010.

[86] S. Venugopal, K. Nadiminti, H. Gibbins and R. Buyya, Designing a Resource Broker for Heterogeneous Grids, Software: Practice and Experience, Volume 38,

Issue 8, Pages: 793-825, ISSN: 0038-0644, Wiley Press, New York, USA, July 10, 2008.

[87] O. Waldrich, P. Wieder and W. Ziegler, A Meta-scheduling Service for Co-allocating Arbitrary Types of Resources, In Parallel Processing and Applied Mathematics, LNCS, Volume 3911/2006, pp. 782-791, 2006.

[88] J. Yu, R. Buyya, "A taxonomy of workflow management systems for grid computing", Journal of Grid Computing, Vol. 3, pp. 171-200, 2005.

Web references

[89] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification", Internet, http://www.ogf.org/documents/GFD.107.pdf, March 2007.

[90] Austrian Grid initiative, http://www.austriangrid.at/, December 2010.

[91] Enabling Desktop Grids for e-Science, http://edgi-project.eu/, December 2010.

[92] Enabling Grids for E-sciencE (EGEE) project, http://www.eu-egee.org/, September 2010.

[93] European Grid Infrastructure, http://www.egi.eu/, September 2010.

[94] EUAsiaGrid project, http://www.euasiagrid.org, September 2010.

[95] European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies (CoreGrid Project), http://www.coregrid.net, December 2010.

[96] I. Foster et al., "OGSA Basic Execution Service Version 1.0", GFD.108, http://www.ogf.org/documents/GFD.108.pdf, November 2008.

[97] GEANT project, http://www.geant.net, September 2010.

[98] Global Grid User Support, http://www.ggus.org, December 2010.

[99] Globus Toolkit, http://www.globus.org/toolkit, September 2008.

[100] gLite middleware for Grid computing, http://glite.cern.ch/, December 2010.

[101] GRIA service-oriented infrastructure, http://www.gria.org/, June 2009.

[102] Grid aided computer system for rapid anti-cancer drug design, CancerGrid project, http://cancergrideu.w3h.hu/, December 2010.

[103] Grid Application Support Centre, http://www.lpds.sztaki.hu/gasuc/, December 2010.

[104] Grid-enabled Know-how Sharing Technology Based on ARC Services and Open Standards (KnowARC) project, http://www.knowarc.eu, November 2009.

[105] GridWay documentation, http://www.gridway.org/doku.php?id=documentation, September 2010.

[106] Grid Workflow Forum, http://www.gridworkflow.org/, September 2008.

[107] The Grid Workloads Archive, http://gwa.ewi.tudelft.nl, September 2009.

[108] Y. Gurevich, "Draft of the ASM Guide", University of Michigan EECS Department Technical Report CSE-TR-336-97, 1997, ftp://www.eecs.umich.edu/groups/gasm/guide97.pdf, September 2009.

[109] The HPC-Europa Project, http://www.hpc-europa.org, September 2009.

[110] HunGrid virtual organisation, http://www.grid.kfki.hu/hungrid/, September 2009.

[111] Information Technology Vocabulary, Fundamental Terms (ISO/IEC 2382-01, 1993), http://jtc1sc36.org/doc/36N0646.pdf, December 2010.

[112] Job Scheduling Hierarchically (JOSH), http://gridengine.sunsource.net/josh.html, October 2007.

[113] Job Submission Description Language (JSDL), http://www.ggf.org/documents/GFD.56.pdf, September 2008.

[114] KnowARC project deliverable no. D3.1-1, "Interoperability Minimal Service Survey", 2007, http://www.knowarc.eu/documents/Knowarc_D3.1-1_07.pdf, September 2010.

[115] The KOALA Co-Allocating Grid Scheduler, http://www.st.ewi.tudelft.nl/koala/, September 2010.

[116] Latin American Grid Program, http://latinamericangrid.org/, September 2010.

[117] LCG-2 User Guide, https://edms.cern.ch/file/454439/2/LCG-2-UserGuide.html, August, 2005.

[118] Molecular Simulation Grid (MoSGrid), http://www.mosgrid.de/, December 2010.

[119] The myExperiment Project workflows, http://www.myexperiment.org/workflows, December 2010.

[120] National Grid Service (NGS), http://www.ngs.ac.uk/, September 2010.

[121] NextGRID – Architecture for Next Generation Grids project, http://www.nextgrid.org/, December 2010.

[122] NorduGrid Middleware, http://www.nordugrid.org/middleware/, September 2010.

[123] Open Grid Services Architecture (OGSA) specifications, http://www.globus.org/ogsa/, September 2010.

[124] Open Grid Forum (OGF), http://www.ogf.org, September 2008.

[125] Open Grid Forum Grid Interoperation Now Community Group, http://forge.gridforum.org/sf/projects/gin, September 2010.

[126] OpenLDAP API, http://www.openldap.org, September 2008.

[127] Open Science Grid (OSG) project, http://www.opensciencegrid.org, September 2010.

[128] The Oxford Advanced Learner's Dictionary, http://www.oxfordadvancedlearnersdictionary.com/, September 2010.

[129] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/, September 2008.

[130] PBS GridWorks, http://www.pbsgridworks.com/, September 2008.

[131] Pacific Rim Application and Grid Middleware Assembly (PRAGMA) Grid, http://www.pragma-grid.net/about/, September 2010.

[132] P-GRADE Portal, http://portal.p-grade.hu/?m=installations&s=0, September 2010.

[133] Phaser experiment, http://goc.pragma-grid.net/wiki/index.php/Run_-Phaser_-on_PRAGMA_grid_and_OSG, September 2010.

[134] ProSim Project/JISC Engage Program, https://sites.google.com/a/staff.westminster.ac.uk/engage/, September 2010.

[135] Sensitivity of the Australian Monsoon to Savannah Fire (Savannah) experiment, http://goc.pragma-grid.net/wiki/index.php?title=Savannah, September 2010.

[136] Semantic Grid Vision, http://www.semanticgrid.org/vision.html, September 2008.

[137] Software Services and Systems Network European Network of Excellence FP7 project, http://www.s-cube-network.eu/, September 2010.

[138] SHaring Interoperable Workflows for large-scale scientific simulations on Available DCIs (SHIWA) project, http://liferay.lpds.sztaki.hu:8080/web/shiwa, December 2010.

[139] Southern Eastern European GRid-enabled eInfrastructure Development (SEE-GRID), http://www.see-grid.org, September 2010.

[140] Success Abandonment Classification workflow by Andrea Wiggins, http://www.myexperiment.org/workflows/140.html, December 2010.

[141] Swiss Grid portal, http://alprose01.projects.cscs.ch:8080/gridsphere/gridsphere, September 2010.

[142] TeraGrid Advanced User Support (AUS) project, https://www.teragrid.org/web/user-support/aus_projects, September 2010.

[143] Uniform Interface to Computing Resources (UNICORE) project, http://www.unicore.eu, September 2010.

[144] UniGrids (former GRIP) Project, http://www.unigrids.org/, September 2008.

[145] Virtual Organisation for Central Europe (VOCE), http://egee.ces-net.cz/en/voce, September 2010.

[146] Westminster Grid Application Support Service (W-GRASS), http://wgrass.wmin.ac.uk, September 2010.

[147] Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Interoperability, September 2010.

[148] Worldwide LHC Computing Grid project, http://lcg.web.cern.ch/LCG/, September 2010.

Publications

[P1] A. Kertész, "Brokering solutions for Grid middlewares", In Pre-proc. of 1st Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, (MEMICS 2005), Znojmo, Czech Republic, 14-17 October, 2005.

[P2] A. Kertész, G. Sipos, P. Kacsuk, "Brokering Multi-Grid Workflows in the P-GRADE Portal", In Euro-Par 2006: Parallel Processing, CoreGRID Workshop on Grid Middleware, Springer-Verlag LNCS, Volume 4375, pp. 138-149, June 2007.

[P3] A. Kertész, P. Kacsuk, "Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service", In Euro-Par 2006: Parallel Processing, CoreGRID Workshop on Grid Middleware, Springer-Verlag LNCS, Volume 4375, pp. 112-115, June 2007.

[P4] A. Kertész, P. Kacsuk, "A Taxonomy of Grid Resource Brokers", In Distributed and Parallel Systems, Springer US, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS'06), pp. 201-210, May 2007.

[P5] A. Kertész, G. Sipos, P. Kacsuk, "Multi-Grid Brokering with the P-GRADE Portal", In Post-Proceedings of the Austrian Grid Symposium (AGS'06), pp. 166-178, OCG Verlag, Austria, 2007.

[P6] A. Kertész, "Grid Brókerek evolúciója: Egységben az erő", Híradástechnika, Volume LXII, pp. 21–25, 2007/12.

[P7] A. Kertész, "The evolution of Grid Brokers: Union for Interoperability", Journal of Scientific Association for Infocommunications with co-operation with the National Council of Hungary for Information and Communications Technology, pp. 55-59, Volume LXIII, HU ISSN 0018-2028, January 2008.

[P8] A. Kertész, P. Kacsuk, "Meta-Broker for Future Generation Grids: A new approach for a high-level interoperable resource management", In Grid Middleware and Services: Challenges and Solutions, 2nd CoreGRID Workshop on Grid Middleware, Springer US, pp. 53-63, June 2008.

[P9] A. Kertész, I. Rodero, F. Guim, "Data Model for Describing Grid Resource Broker Capabilities", In Grid Middleware and Services: Challenges and Solutions, 2nd CoreGRID Workshop on Grid Middleware, Springer US, pp. 39-52, June 2008.

[P10] A. Kertész, I. Rodero, F. Guim, "Meta-Brokering approaches in state-of-the-art Grid Resource Management", CoreGRID Integration Workshop 2008 – Integrated Research in Grid Computing, pp. 371-382, Hersonissos, Crete, Greece, April 2008.

[P11] A. Kertész, Z. Farkas, P. Kacsuk, T. Kiss, "Grid Interoperability by Multiple Broker Utilization and Meta-Brokering", In Grid Enabled Remote Instrumentation, Springer US Book Series on Signals and Communication Technology, (INGRID'07), pp. 303-312, October 2008.

[P12] P. Kacsuk, A. Kertész and T. Kiss, "Can We Connect Existing Production Grids into a World Wide Grid?", In High Performance Computing for Computational Science (VECPAR'08), Springer LNCS, Volume 5336, pp. 109-122, December 2008.

[P13] A. Kertész, J. D. Dombi, J. Dombi, "Adaptive scheduling solution for grid meta-brokering", Acta Cybernetica, Volume 19, pp. 105-123, 2009.

[P14] A. Kertész, I. Rodero, F. Guim, "Meta-Brokering Solutions for Expanding Grid Middleware Limitations", In Euro-Par 2008 Workshops – Parallel Processing, Workshop on Secure, Trusted, Manageable and Controllable Grid Services (SGS'08), Springer LNCS, Volume 5415, pp. 199-210, April 2009.

[P15] A. Kertész, G. Kecskeméti, I. Brandic, "An SLA-based Resource Virtualization Approach For On-demand Service Provision", In proceedings of 3rd International Workshop on Virtualization Technologies in Distributed Computing (VTDC'09) in conjunction with ICAC'09, Barcelona, Spain, ACM, pp. 27-34, June 15, 2009.

[P16] A. Kertész and Zs. Németh, "Formal Aspects of Grid Brokering", In EPTCS 14, 8th International Workshop on Parallel and Distributed Methods in verifiCation (PDMC'09), pp. 18-31, CoRR abs/0912.2549, 2009.

[P17] A. Kertész, P. Kacsuk, "Grid Interoperability Solutions in Grid Resource Management", IEEE Systems Journal's Special Issue on Grid Resource Management, Volume 3, Issue 1, pp. 131-141, March 2009.

[P18] A. Kertész and T. Prokosch, "The Anatomy of Grid Resource Management", In book: Remote Instrumentation and Virtual Laboratories, Eds.: Davoli, F.; Meyer, N.; Pugliese, R.; Zappatore, S., Springer Science+Business Media, LLC, pp. 123-132, 2010.

[P19] A. Kertész, P. Kacsuk, "GMBS: A New Middleware Service for Making Grids Interoperable", Future Generation Computer Systems, vol. 26, no. 4, pp. 542-553, 2010.