European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management

A. Kertész, P. Kacsuk
{attila.kertesz, kacsuk}@sztaki.hu
MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P.O. Box 63, Hungary

I. Rodero, F. Guim, J. Corbalan
{irodero, fguim, juli}@ac.upc.edu
Technical University of Catalonia (UPC)
Jordi Girona 1-3, 08034 Barcelona, Spain

CoreGRID Technical Report
Number TR-0116
November 13, 2007

Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management

A. Kertész, P. Kacsuk
{attila.kertesz, kacsuk}@sztaki.hu
MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P.O. Box 63, Hungary

I. Rodero, F. Guim, J. Corbalan
{irodero, fguim, juli}@ac.upc.edu
Technical University of Catalonia (UPC)
Jordi Girona 1-3, 08034 Barcelona, Spain

CoreGRID TR-0116
November 13, 2007

Abstract

Grid resource management is probably the research field mostly affected by user demands. Though well-designed, evaluated and widely used resource brokers, meta-schedulers have been developed, new capabilities are required, such as agreement and interoperability support. Existing solutions cannot cross the border of current middleware systems that are lacking the support of these requirements. In this paper we examine and compare different research directions followed by researchers in the field of Grid Resource Management regarding the level on top of brokering approaches. Our proposed meta-brokering approach means a higher level resource management by enabling communication among existing Grid Brokers and utilizing them. We state the requirements of this novel middleware service, and define a general meta-brokering architecture to be implemented as a Service Oriented Knowledge Utility (SOKU) in future generation grids.

1 Introduction

More than a decade has passed and Grid Computing has become a detached research field targeted by many worldwide projects. Several years ago users and companies having computation and data intensive applications looked skeptical at the forerunners of grid solutions, who promised less execution times and easy-to-use application development environments. Research groups were forming around specific middleware components, and different research branches has grown out from the trunk. Many user groups from various research fields put their trust in grids, and today usage statistics and research results show that they were undoubtedly right. Nowadays research directions are focusing on user needs; more efficient utilization and interoperability play the key roles. Grid resource management is probably the research field mostly affected by user demands. Though well-designed, evaluated and widely used resource brokers, meta-schedulers have been developed, new capabilities are required, such as agreement and interoperability support. These two directions also depend on other grid middleware capabilities and services, and since they hardly can cross the border of these middleware solutions, they need revolutionary changes affecting the whole system. Trying to enlarge the limitation borders, in this paper we introduce a meta-brokering approach that enables a step-by-step evolution, and does not need radical changes to the whole system.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

The need for interoperability among different grid systems has raised several questions and directions. The advance of grids seems to follow the way envisaged and assigned by the Next Generation Grids Expert Group [1]. The SOKU architecture [1] enables more flexibility, adaptability and advanced interfaces, therefore interoperability is evident and congenital in these systems. Moving towards this direction two works envisaged WWG (World Wide Grid) as the next generation of grids in a similar manner as the Internet was born. The first vision is the InterGrid [4], which promotes interlinking of grid systems through peering agreements to enable inter-grid resource sharing. This approach is a more economical view, where business application support is a primal goal, and this also supposed to establish sustainability. Building this global cyber-infrastructure would require current grid systems to adopt mechanisms that enable administrative separation by allowing network of networks. This would mean supporting: peering policies for sharing, provisioning and allocating resources, new application models that can cope with higher dynamicity, virtual organizations spanning multiple grids, and finally enabling inter-grid resource brokering. In this new 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. Its main task is to manage peering agreements and discover and allocate resources (from different domains). Though we can identify a higher level resource management approach in this vision, the proposed solution still searches and maps resources to user requests and no meta-data is consumed regarding intra-grid brokers. The second approach envisages WWG [5] to be reality within 1-2 years. This work states that five major steps are required to create the next generation architecture: existing production grids should be connected by *uniform meta-brokers*, *workflow-oriented portals should interface* these meta-brokers, a *repository of workflow grid services* should be created, a *new security concept should be designed for trustable dynamic VOs*, and finally a *grid marketing model* should be created. The later requirements have also appeared in the InterGrid approach – this seems to require more time to be realized. As a final solution this paper supposes that inter-connected meta-brokers should solve the interoperability among different grids through uniform interfaces. Both visions proceed from the current grid architectures and conclude in a more or less redesigned one.

In the following sections of this paper we are focusing only on grid resource management and present the state of the art in meta-brokering approaches and research directions. In Section 2, we name the different acronyms and expressions used by existing solutions and clarify their meaning and differences. In Section 3, we describe meta-level research directions and propose a general meta-brokering solution after analyzing its requirements. At the end of Section 3, we show how the presented architecture can be realized in two different grid user environments, finally we state how a future version can act as a heart of WWG, and conclude the paper in Section 4.

2 Definitions of Grid Resource Management Tools

Before we take a closer look on current research directions towards next generation solutions in Grid Resource Management, we need to clarify definitions of current solutions. Searching and comparing related works we meet with meta-schedulers, local and global schedulers, brokers, resource allocation managers and so on. In this section we gather and classify the expressions used in the area of Grid Resource Management. We refer to these definitions further on in this paper by naming specific components of grid middleware. On Figure 1, we can see the architecture of a Grid System – focusing on resource utilization.

In the following we define the acronyms used in this figure, and name the generally used synonyms or similar expressions:

- $\mathcal{R}$ – resource: In general it means a physical machine, where user programs are executed. We distinguish between two types regarding utilization: Computing Element (CE), where the user program (also task or job) execution takes place, and Storage Element (SE), where user data is stored. Web Services (WS) or other Grid Services (GS) are also regarded as resources, since they similarly produce output for a given user input. Finally Remote Instruments (RI or IE as Instrument Element [6]) are various tools to access, process, produce or consume data usually for physicists, chemists, biologists or other researchers.

- $\mathcal{LRMS}$ – local resource management system, scheduler, local scheduler, local resource manager, sub-scheduler: These tools are usually cluster managers that were taken from high-performance and distributed computing, and now generally used in Grid Systems. More information on the most widespread ones (LSF, PBS, SGE) can be found in a survey in [7].

- $\mathcal{RAM}$ – resource access manager: This is usually a component of a grid middleware that accesses the resource

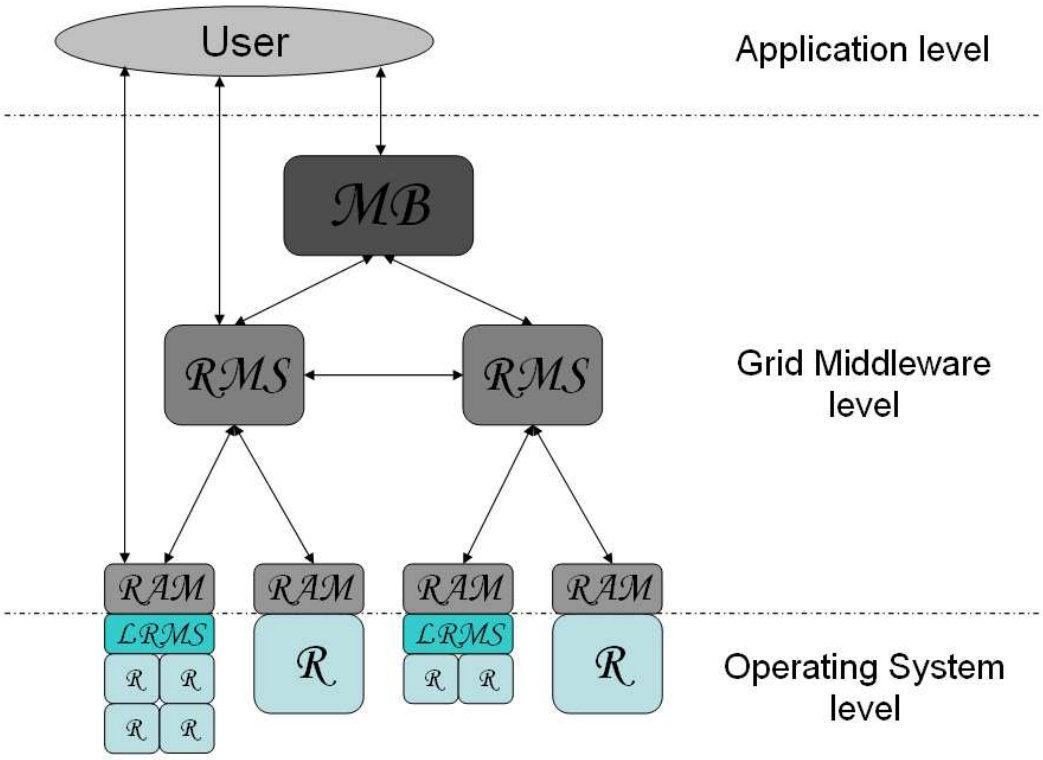

Figure 1: Resource managing components in Grid Systems

and handles the arriving jobs. It provides an interface for the middleware to the resource. An example is GRAM in the Globus Toolkit [8].

- $\mathcal{RMS}$ – resource management system, meta-scheduler, global scheduler, resource broker, grid scheduler, orchestrator: It consists of one or more components of a grid middleware. It can also be an external tool that uses middleware components, services. Later we will investigate these tools more detailed.

- $\mathcal{MB}$ – meta-broker, inter-grid broker, inter-grid gateway: It is a novel higher level brokering service. It sits on top of the resource brokers and uses meta-data of them to decide where to send a user job. Section 3 deals with this component in details.

Though the named expressions of grid resource managers denoted by RMS are usually inter-changeable, they slightly differ. Figure 2 is used to reveal these differences. In general a meta-scheduler is focusing more on job scheduling (executing an algorithm), a resource 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 a middleware to be able to manage the whole job life-cycle. In this sense scheduling is an atomic process, brokering incorporates scheduling, and resource management includes brokering. A taxonomy and survey of resource brokers can be found in [3], and another one of grid resource management systems in [2].

Up to the level of resource management systems current grids are well studied, reliable enough to use this architecture and serve user communities. The remaining part between the users and the brokers, the meta-brokering layer is under development. The need for it has already been identified by different research groups. This missing level is supposed to establish interoperability among different grid systems, and provide a common grid access method for users. The next section shows the current state of the art.

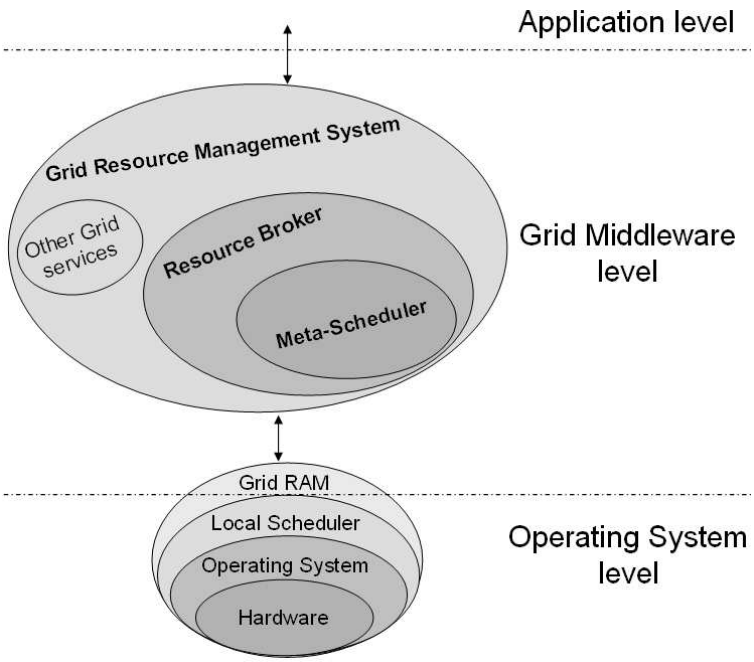

Figure 2: Current Grid Resource Manager Anatomy

3 Interoperability efforts with higher level Resource Management

3.1 Enabling inter-broker communication, data-flow among Brokers

The current solutions of grid resource management will not be able to fulfill the high demands of future generation grid systems. Several research groups have noticed this problem and have started to look for solutions to enhance interoperability. One of the promising approaches aimed at enabling communication among existing resource brokers. This subsection introduces the work done in this direction.

The GSA-RG of OGF[1] is currently working on a project enabling grid scheduler interaction. They try to define common protocol and interface among schedulers enabling inter-grid usage. To achieve this, they use standard tools (JSDL[2], OGSA[3], WS-Agreement[4], and propose an SDL (Scheduling Description Language) to extend the currently available job description language (in the next subsection we deal with SDL more detailed). This work is still in progress, up to now only a partial SDL schema has been created. Lately researchers of this group were paying more attention to agreements, and in a recent CoreGRID technical report they describe using WS-Agreement to make negotiations and perform interaction [9].

Another instance of this approach is the Latin American Grid initiative (LA Grid[5]), 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 enablement, autonomic resource management, meta-brokering 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 meta-broker instances with a set of functional modules: connection management, resource management, job management and notification management. These modules implement the APIs and protocols used in LA Grid through web services. A meta-broker instance interacts with existent brokers within the resource domain. In LA Grid, resources of different institutions belong to their respective resource domains and each resource domain has a meta-broker instance. Inside a domain, a

[1] https://forge.gridforum.org/sf/projects/gsa-rg
[2] http://www.ggf.org/documents/GFD.56.pdf
[3] http://www.globus.org/ogsa/
[4] http://www.ogf.org/documents/GFD.107.pdf
[5] http://latinamericangrid.org/

meta-broker instance controls resources directly and/or indirectly through other brokers in the domain. In the former case, each resource reports its information directly to the broker instance using resource management web services. In the latter case, an existent broker is responsible for reporting the information of the resources under its control. The system works in the following way: each meta-broker instance collects resource information from its neighbors and save the information in its resource repository or in-core memory. 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. Example of aggregated resource information on a set of servers in a domain is: ProcSpeed={(1200-3000,<count=5>,<total=19000>)}; OSType={(Linux,<count=5>)}. A job request need to be described in JSDL and 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. One of the factors 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 the meta-broker instance in another domain. From that point this meta-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[6] was designed to work on DAS-2 interacting with Globus [8] middleware services with the main features of data and processor co-allocation. Lately it is being extended to support DAS-3 and Grid'5000 [10]. To inter-connect 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. In an ongoing experiment they use a so-called delegated matchmaking (DMM), 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[7] has also broadened its support for multiple grids. In their latest paper they introduce a Scheduling Architectures Taxonomy [11], 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 DMM. Gridway is also based on Globus, and they are experimenting with GT4[8] and gLite[9].

Comparing the previous approaches, we can see that all of them use a new methodology 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. Usually the local domain is preferenced, and when a job is passed to somewhere else, the result should be passed back to the initial point. The used peer-2-peer approach seems to provide a good performance, but if we wanted to apply the presented methods to different systems, domains, we are facing interoperability problems, again. This leads us back to the work done by OGF to define common interfaces. Since it takes much time to agree on these protocols and adopt them, in the next subsection we focus on a solution that utilizes and delegates broker information by reaching the brokers through their current interfaces. We gather and utilize meta-data about existing widely used brokers from various grid systems to establish a real meta-brokering infrastructure.

3.2 Introducing a meta-level above Resource Brokers

As we mentioned in the introduction, the NGG Expert Group has identified a convergence between grid and web services [1]. 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 SOKUs, which will become the building blocks of future grids. This convergence of grid services bring 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, we 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.

[6] http://www.st.ewi.tudelft.nl/koala/
[7] http://www.gridway.org/
[8] http://www.globus.org/toolkit/
[9] http://glite.web.cern.ch/glite/

Our research targeted the area of grid resource management, and builds upon agent technologies and semantics. We introduce a novel scheduling philosophy called meta-brokering that creates a meta-level above current resource management solutions by using technologies from the area of the semantic web as stated in the semantic grid vision[10]. Following this way, we have developed solutions to make data about resource managers available for cooperated, automatic processing in the form of meta-data. We provide language schemas to store and share this meta-data, and to be processed by various scheduling policies. In this subsection, first we examine the requirements of the meta-brokering approach, then present a general solution that can be realized in different grid environments shown in the second part. Two approaches follow this meta-brokering direction. Both projects identified the need for an automatic, intelligent way for selecting a proper domain, VO or grid managed by resource brokers.

The first one is the HPC-Europa project[11], which aims 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, local policies, and so on. For instance, there are currently five different systems that provide job submission and basic monitoring functionality in the HPC-Europa infrastructure: eNANOS [15], GRIA middleware[12], Grid Resource Management System (GRMS)[13], Job Scheduling Hierarchically (JOSH)[14] and UNICORE[15]. 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 analyzing 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 in a straightforward manner by building portlets. Using these interfaces the accompanying resource managers can build a plugin-based component. From the end-user perspective, a uniform Graphical User Interface is provided that is common for all systems deployed in the HPC-Europa infrastructure. This GUI can be dynamically adapted to particular systems and still keep the same look and feel. 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).

The second approach is the P-GRADE Portal [14], which is also 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, analyzing 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[16], GTbroker [12] and the NorduGrid broker[17]. 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 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 the previous subsection we saw a possible solution, where brokers pass the job to another resource manager instance of a different domain, if the actual is overloaded. The meta-brokering approach creates a general view of the available brokers and resources. This solution enables the higher level meta-broker to have a global, up-to-date view of capabilities and availability. In the following we state the

[10] http://www.semanticgrid.org/vision.html
[11] http://www.hpc-europa.org/
[12] http://www.gria.org/
[13] http://www.gridlab.org/grms
[14] http://gridengine.sunsource.net/josh.html
[15] http://www.unicore.eu/
[16] http://glite.web.cern.ch/glite/
[17] http://www.nordugrid.org/middleware/

requirements of such a general meta-broker and introduce its architecture.

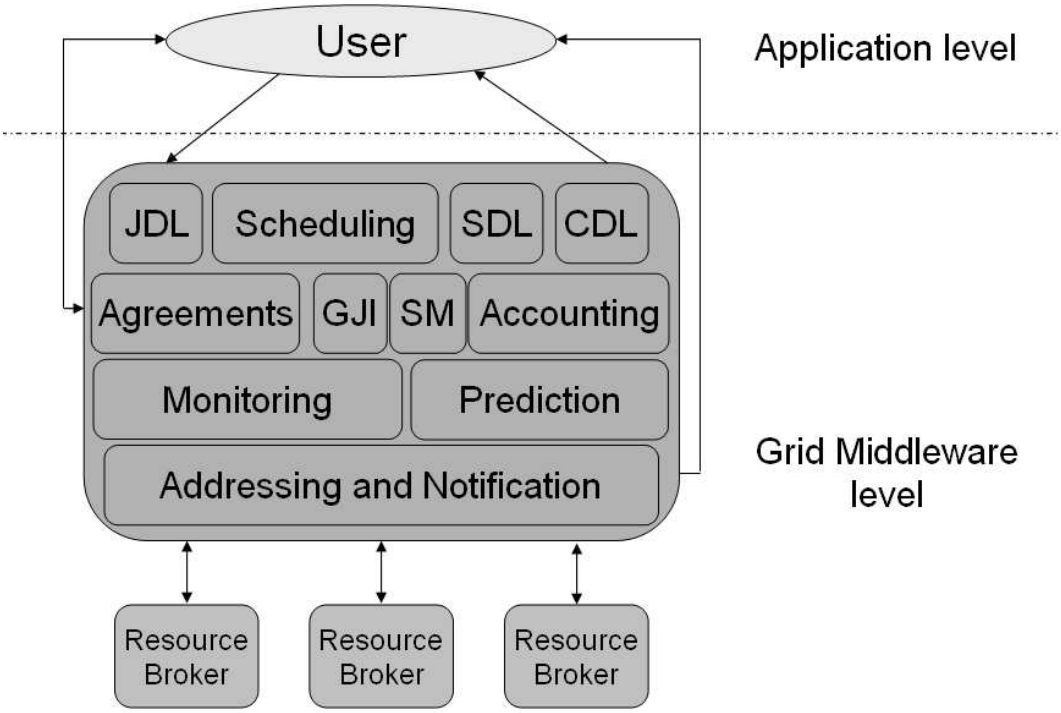

Figure 3: Required components of a General Meta-Broker

Figure 3 is intended to show all the components needed by a Meta-Broker. In the following we describe these components by introducing the main requirements of this higher level brokering service:

- JDL – Job Description Language: Since the goal of this service is to offer a uniform way to access various grids, a unified description format is needed to specify all the user requirements.
- CDL – Capability Description Language: Each broker has 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 Capability Description Language. It should be general enough to include all the service capabilities (regarding interfaces, job submission, monitoring and agreements).
- SDL – Scheduling Description Language: Besides CDL and JDL the scheduling requirements and policies also need to be stored. 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.
- Matchmaking Mechanism: This component performs the scheduling 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. Providing a transparent way for users these various proxies also need to be handled by the Meta-Broker. It is obvious that only those grids and brokers can be considered for job submission, to which the user has a valid certificate.

- Accounting Mechanism: The GJI and PM 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.
- Agreements Mechanism: This component is in connection with the Accounting mechanism. Service Level Agreements (SLA) are planned to be used in future generation grids. 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 the brokers.
- Monitoring Mechanism: Reliable operation requires global monitoring, in terms of the inter-connected brokers, reachable domain, grid resources and loads, and local component functionalities. Self-aware and fault tolerant operation need to be provided by the system itself, which needs extensive monitoring.
- Prediction Mechanism: This component is in connection with the Monitoring and Matchmaking mechanisms. It is necessary to perform calculations of broker availability, service utilization and user request loads 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 even and external job state notifications.

As we stated in the beginning of this section, semantics are crucial to establish interoperability. Standardization should be taken into account during the design and development of sustainable solutions. To walk on this way, we use the standards and widespread technologies, where applicable. Though the requirements and the General Meta-Broker Architecture meant to be implementation and technology independent we recommend the use of JSDL (Job Submission Description Language)[18] as JDL for describing user jobs, and WS-Agreement for handling agreements. Regarding CDL we developed a language called BPDL (Broker Property Description Language) [17], which was also incorporated SDL. In the rest of this subsection we revise and upgrade BPDL and gather the scheduling-related attributes to MBSDL (Meta-Broker Scheduling Description Language). To contribute to the standardization process of OGF (Open Grid Forum), we keep on negotiating with the OGF-GSA research group to create a standard SDL. Once it is finalized, it will be used instead of MBSDL.

During the clarification and separation of BPDL and MBSDL attributes, we used the same data model definition as in [17]. It implies that the structure of the new BPDL – that we call BPDL 2.0 –, remains nearly the same, we have only clarified some attributes, added wanting ones and separated the scheduling-related ones to MBSDL. As an implementation of the data model, we used XML schemas again, in both language specifications. Figure 4 and 5 shows these schemas regarding BPDL 2.0 and MBSDL.

Using these schemas, we are able to store metadata about every resource broker. BPDL 2.0 is used for storing static capabilities and dynamic states of the utilized brokers. The static information is represented by the following fields:

- BrokerID: It contains a unique identifier of a resource broker.
- Interface: This field provides metadata about the accessibility and notification methods of the broker.
- 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.
- Other static information can also be stored by using the attributes of MBSDL described later. The most important ones 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 should be updated by the Meta-Broker 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 shows, how successfully the broker performed job requests, and how reliable its services are. The Prediction attribute can be used to store predicted data about broker availability and reliability.

[18] http://www.ogf.org/documents/GFD.56.pdf

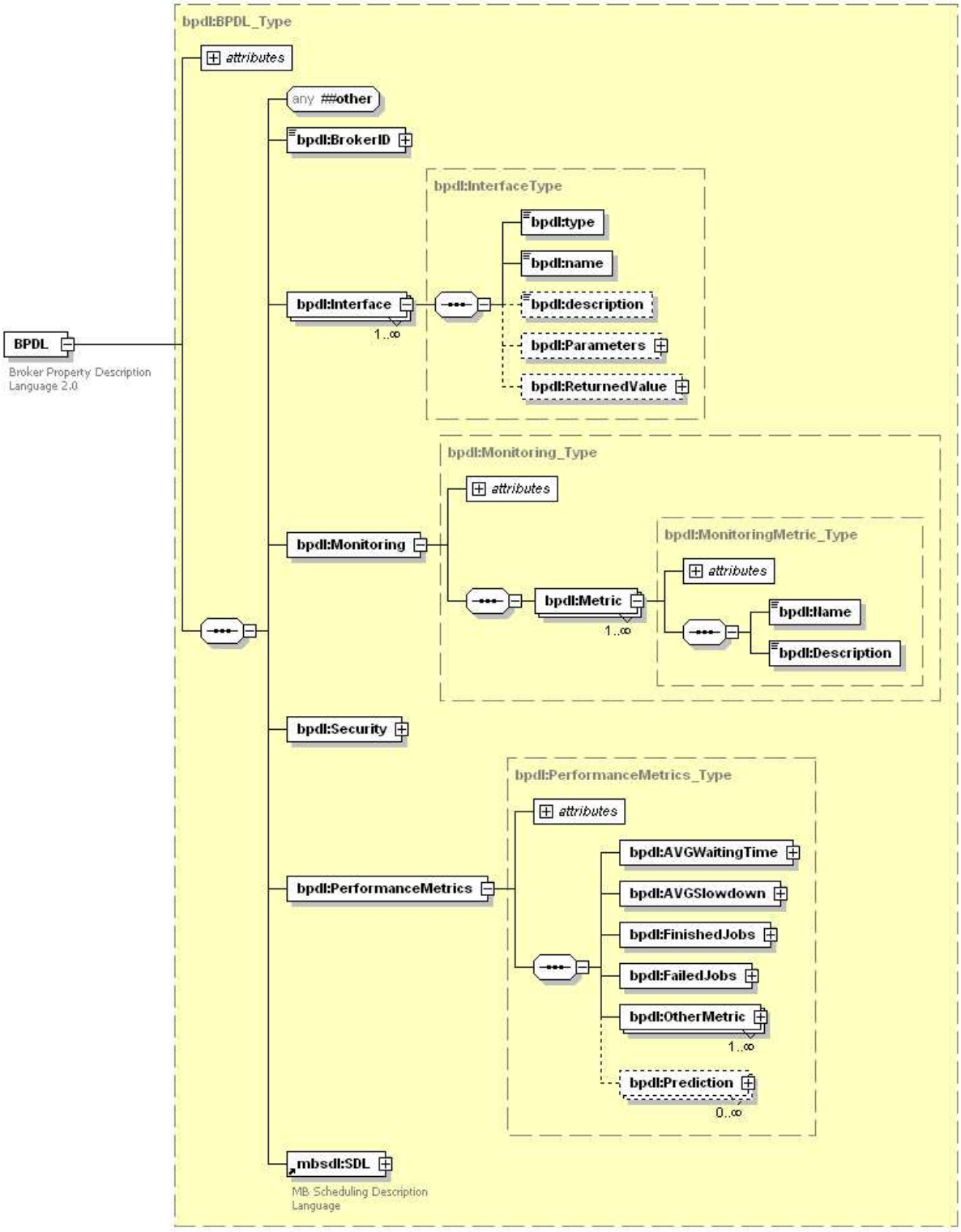

Figure 4: General XML schema of BPDL 2.0

The MBSDL language can be used to extend BPDL with scheduling related attributes. Since JSDL is also lacking these attributes, MBSDL can also be regarded 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 tolerance features can also be selected to affect the schedule.
- Policy: In this filed we can choose from various scheduling policies, or we can define customized ones. The LRMSPolicy is used to describe local scheduler capabilities.

Having all the requirements defined, we are able to implement and build a meta-brokering service. In the following, we show two realization of the presented meta-brokering architecture in two widely used grid portal environments.

The first solution, the Grid Meta-Broker, will be used to solve meta-brokering in the future version of P-GRADE Portal (Figure 6). Nevertheless it has been designed as a standalone, standards-base Web Service, therefore it is grid middleware and portal independent. As JSDL is general enough to describe jobs of different grids and brokers, this language has been chosen to be the job description language of the Grid Meta-Broker. The Translator component of the Meta-Broker is responsible for translating the resource specification language defined by the user to the language of the appropriate resource broker that the Grid Meta-Broker selects to use for a given job. From all the various job specification languages 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 solution. Even the same methodology can be expressed in different ways in different languages. Therefore it is really important to use a general scheduling language. This is the MBSDL, so it should be used by the users to specify these special attributes. Besides describing user jobs, we also need to describe resource brokers in order to manage and make difference among them. These brokers have various features for supporting different user needs. To achieve this, the Grid Meta-Broker uses the above described BPDL together with MBSDL. The Information Collector (IC) component of the Meta-broker stores the data of the reachable brokers and historical data of the previous submissions in BPDL. 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 performance attribute in the BPDL of the appropriate broker. In this way, these documents represent and store the dynamic states of the brokers. The load of the resources behind the brokers is also taken into account to help the MatchMaker component to select the proper environment for the submitted job. When too many similar jobs are needed to be handled by the Meta-Broker the so-called best effort matchmaking may flood a broker and its grid. In order to cope with this problem, there are IS Agents reporting to the Information Collector, which regularly check the load of the underlying grids of each connected resource broker, and store this data. They are in connection with the Information System of the grids behind the utilized brokers. With this additional information the matchmaking process can adapt to the load of the utilized grids. The actual state (load, configurations) of the Grid Meta-Broker is also stored here, and it can also be queried by users. The previously introduced language attributes are used for matching the user requests to the description of the interconnected brokers: which is the role of the MatchMaker component. The matchmaking process relies on one of the pre-defined scheduling policies. We investigate these policies in details later in section 3.4. To help the MatchMaker, the IS Agents monitor the interconnected grids, and provide load and resource availability information. The actual performances of the brokers can also be useful, this data is stored and updated regularly PerformanceMetric field of the BPDL instances. The Invokers are broker-specific components: they communicate with the interconnected brokers, invoking them with job requests and collecting the results. Data handling is also an important task of this component. The Invoker instance, responsible for managing certificates and job submission to the actually selected broker, takes care of transferring the necessary files to the selected grid environment. After job submission, it stages the output files back and upgrades the historical data stored in the Information Collector with the logging data of the utilized broker. The Core component of the Meta-Broker is responsible for managing the communication (information and data exchange) among the other components.

On Figure 7 we can see the metabrokering-enabled HPC-Europa SPA. This extended version includes new services to perform meta-brokering: a new module for the broker scheduling, a scheduling plug-in for each center, a language

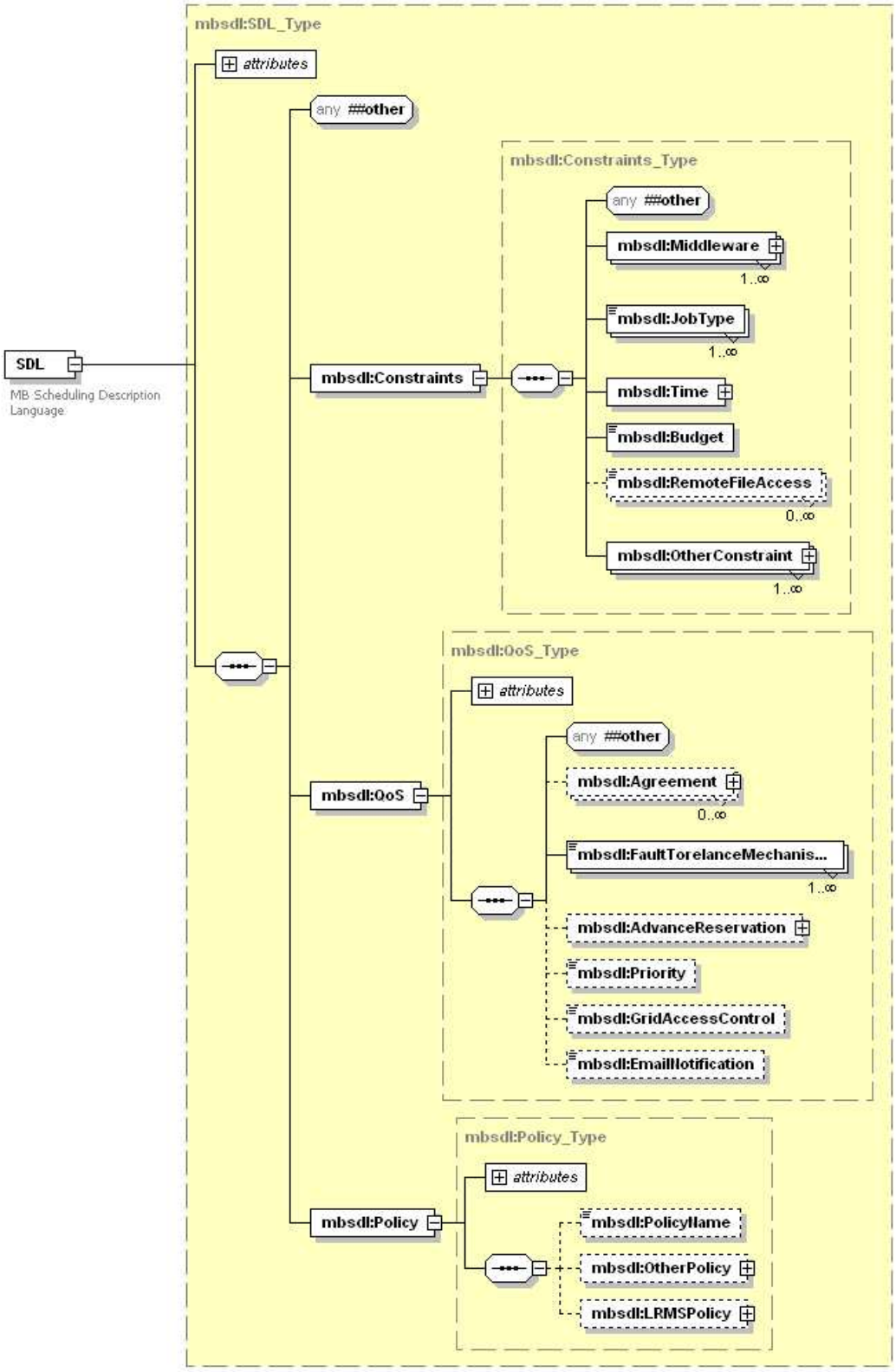

Figure 5: General XML schema of MBSDL

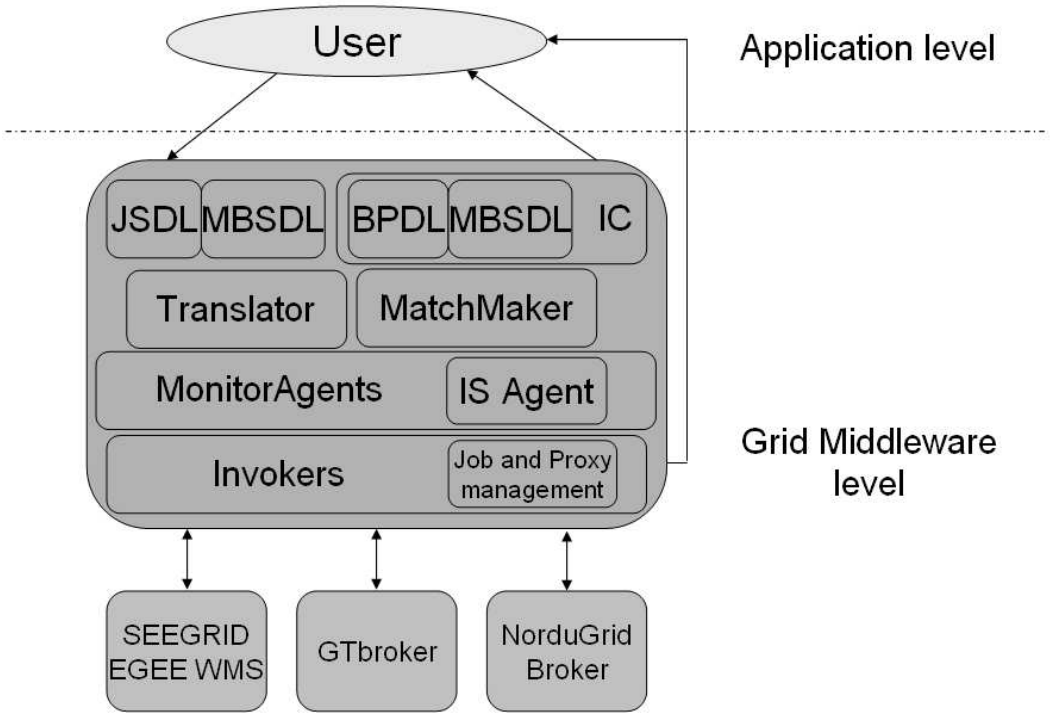

Figure 6: Grid Meta-Broker Architecture in future P-GRADE Portal

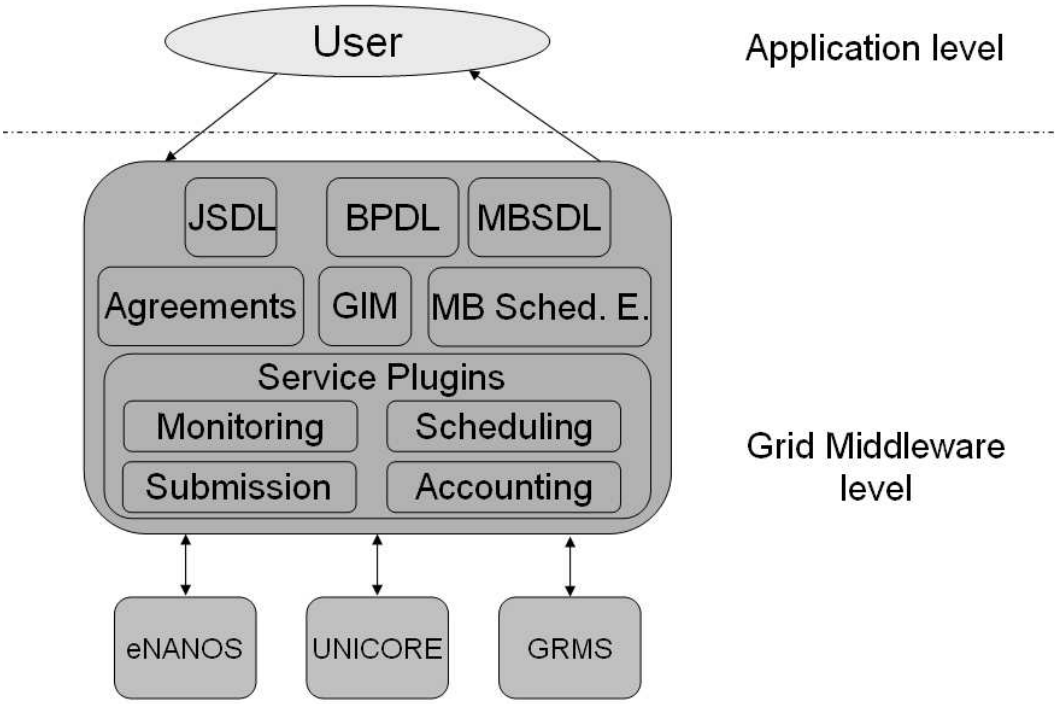

Figure 7: Meta-Broker Architecture in the HPC-Europa SPA

to describe the scheduling capability of brokers and global identifiers management. Moreover, it includes some other services, such as a predictor service or a historic data catalog to improve the scheduling techniques. The idea was to design a system following the OGF standards such as the JSDL as a language for describing jobs. With the help of BPDL and SDL, other useful information can be stored to improve the scheduling strategy. Some of this information can be drawn from previous decisions regarding the selection of brokers as a historical data, such as the achieved quality of service by the different brokers, the average waiting time, the reputation of brokers, or the level of availability and reliability of the resources under the broker domains. In the new architecture, the meta-brokering scheduling engine (MB Sched. E.) is responsible for the broker selection to decide where to submit a job. At the portal level, it evaluates one of the meta-brokering policies available in the framework and in addition to this component a new portlet is be provided to configure policies and its main issues. To map the scheduling performed in the meta-broker scheduling engine following the scheduling capabilities of the different brokers, we needed a new plug-in. This plug-in implements the scheduling API and provides the functionality supported by each brokering system. To define the scheduling capability of the different centers (brokers), we need a scheduling description language. This language is the MBSDL, which includes the scheduling policy capabilities, such as the quality of service, priorities, the support for co-allocations and for advanced reservations, economic issues, or the scheduling policy family. For example, in eNANOS we can offer as a scheduling capability the load-balancing of parallel applications in run-time or the support for MPI+OpenMP applications [15]. Finally, to allow the portal managing jobs coming from all the centers, we need global identifiers mechanisms. To obtain global identifiers, we designed a new service, the global identifier manager (GIM), that is accessible from all the services and through the centers' plug-ins. Different brokers should support this new functionality via their plug-ins. Regarding implementation, the Universally Unique Identifier[19] (UUID) seems to be a good candidate as the identifier standard. It is used in software construction, standardized by the Open Software Foundation (OSF) as part of the Distributed Computing Environment (DCE).

3.3 The marriage of these two approaches

In subsection 3.1, we have seen how efficient systems can be built using a novel approach to enable inter-broker communication and inter-domain resource utilization with peer-2-peer extensions of existing resource managers. In subsection 3.2, we have introduced the meta-brokering approach and a general meta-brokering structure. Finally we have shown how this methodology can be applied in two real-world grid environment. In the introduction, we discussed where grid evolution is heading, the future generation of grids, the WWG. Though implementing the general meta-broker inevitably takes us closer to WWG, in the future it is necessary to use the techniques shown in 3.1. At this meta-level it is not the resource information that should be delegated, but the broker information. Since exactly this data is stored in BPDL and MBSDL, we need to make them available among all the peering instances to create a knowledge base. This final global meta-broker utility (Figure 8.) will be able to build up an extensive view of interoperating grids, and manage them efficiently. Once this goal is accomplished, we will arrive to the WWG vision by creating a service-oriented knowledge utility (SOKU), which was exactly the same aim assigned by the Next Generation Grids Expert Group.

3.4 Scheduling policies in meta-brokering

In addition to the architecture model and the required interfaces, there is an important novel functionality: the scheduling at the meta-brokering level. We can implement different kinds of scheduling policies depending on the information we have at the meta-brokering level. If the meta-broker gathers detailed information about the resources, it can implement the typical scheduling policies used by resource brokers or meta-schedulers to access resources directly. In this case a meta-broker can be regarded as a multi-grid resource broker, and it would provide nothing more than other well-known solutions described in [3]. The novel scheduling policies use meta-information about brokers and resources to cope with larger amount of resources managed by lower level brokers acting as local intra-domain schedulers, job dispatchers and execution entities. Based on the infrastructure shown in Figure 5, we propose meta-brokering policies relying on the capabilities and measured performances of the utilized brokers. In this case, the selection of the appropriate brokering system can be done using a multi-criteria algorithm that can take into account the gathered and predicted metadata stored in SDL and CDL. In our case the general scheduling policy examines the job description, which is stored in JSDL and MBSDL, and matches them with the metadata stored in BPDL and MBSDL. In order to cope with the high dynamicity of the grid domains, it uses measured and predicted data in BPDL. To present a

[19] http://www.ietf.org/rfc/rfc4122.txt

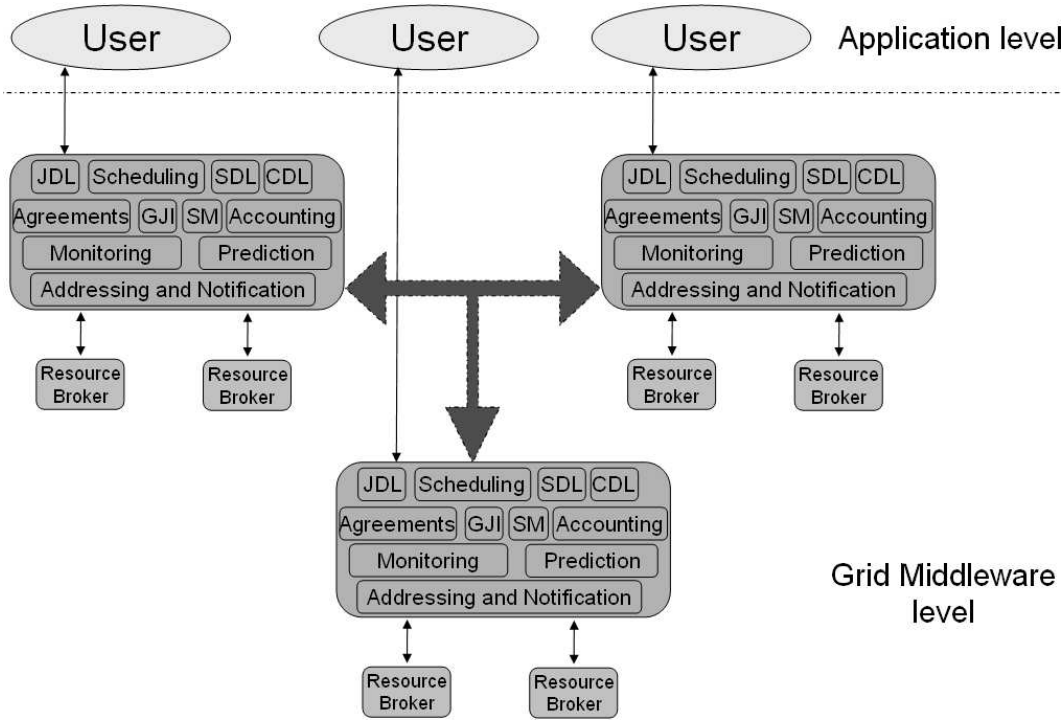

Figure 8: Global Meta-Brokering Service as a SOKU

formalization of this meta-brokering algorithm, we use the same notation and model presented earlier in [17]. We defined the following function for matchmaking:

- μ: $\mathcal{T} \times \mathcal{B}^{\,i} \rightarrow \mathcal{B}$, where $\mathcal{T}$ is a set of tasks (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 is a special element, which is an empty description. This is the return value, when no broker fits the job requirements.)

The algorithm of this function contains the following notations and steps:

$\mathcal{B} = b_0, b_1, ..., b_n$, brokers
$t \in \mathcal{T}$, job
jb: basic static information and constraints in JSDL + MBSDL of t
js: scheduling and QoS requirements in MBSDL of t
$\mathcal{P} = p_1, ..., p_n$, static information and performance data in BPDL + MBSDL of $b_i, 1 \leq i \leq n$
$\mathcal{S} = s_1, ..., b_n$, scheduling policies and capabilities in MBSDL of $b_i, 1 \leq i \leq n$
$\mathcal{R} = r_0, r_1, ..., r_n$, performance ranks
$r_0 = \epsilon > 0$, an arbitrary small fixed number

FOR i = 1 TO n
IF checkJobReq(jb,p_i)
THEN

```
        IF checkSchedReq((js,s_i)
            THEN r_i = countPerfRank(p_i)
            ELSE r_i = countPerfRank(p_i)/10
    ELSE r_i = 0
k = pozOfHighestRank(R)
RETURN b_k
```

The matchmaking algorithm compares the description of the actual job to the meta-data of the registered resource brokers. In the first phase the basic job requirement attributes stored in JSDL, and the constraints stored in the MBSDL extension are matched against the basic broker capabilities stored in BPDL extended with MBSDL: this selection determines a group of brokers that are able to fulfill the job request. In this phase those brokers are kept that are able to submit the specified job type and work with the requested middleware services. In the second phase the brokers are filtered by all the scheduling requirements stated in the MBSDL extension of the user requirements. For all of the brokers that could pass these filtering phases a rank is counted by the actual broker performance and domain load. For the brokers that could only pass the first phase reduced ranks are assigned. Finally the list of the brokers is ordered by these ranks. The broker with the highest rank is selected for submission. This algorithm is used in the meta-brokering architecture presented in Figure 6.

Different scheduling policies stored in MBSDL can affect the scheduling. It depends on the deployed architecture, which policies are implemented in the *checkSchedReq* method. In economy-based grid environments scheduling policies could rely on the agreements and accounting information to match user budget and maximize domain earnings. In this case the selected broker can be used for negotiations and agreeing on service terms. If they cannot agree, the broker with the second highest rank will be selected for starting negotiations. Another approach is based on an ongoing work in UPC to use policies based on historical job/resource information to make scheduling decisions. Data mining techniques are used regarding the historical information of Grid resources usage, Grid workloads, and a minimum set of job requirements (e.g., executable, number of processors and input files) to estimate variables about: (1) the job requirements, which estimate how much processor, disk and memory a job will use; and (2) the future state of the different resources evolved in the Grid, which estimate, for instance, how much free space will be available in a given host, or what load it will have in a given future time. This information is derived by correlating the past executions of similar jobs using similar resources, or with similar future load using similar prediction techniques that have been proposed in other works in terms of job performance prediction and resource usage but using Grid workloads. This kind of scheduling policy can be applied to architectures described in subsection 3.1.

Using the achievements of this latest approach, we can improve the meta-brokering algorithm and create another policy by extending the *countPerfRank* method with estimation and prediction results. But in our case it is not the resource availability that needs to be estimated, but the services, capabilities of brokers and their reliability.

The Global Meta-Brokering Service, shown in Figure 8, requires further extensions and investigations regarding scheduling policies. A future inter-connected meta-brokering architecture would require a further phase (third phase) in its algorithm. In this phase denied job requests (by job failures, broker unavailability or violated agreements) should be sent to another meta-broker instance for submission. This approach opens several questions, such as which instance to choose, when should an agreement be violated or how likely it is to find a better broker for a request in a different instance domain.

4 Concluding remarks

In this paper we have examined and compared different research directions followed by researchers in the field of Grid Resource Management regarding higher level brokering approaches. First we clarified the differences and similarities of currently used solutions in the area of grid resource management by presenting a Grid Resource Manager Anatomy. This model defines the connections and responsibility boundaries of current tools. After introducing the latest trends in grid development and the assigned grid evolvement directions by the Next Generation Grids Expert Group, we have shown how grid resource management can follow this way, and what kind of steps are necessary to be taken to establish a higher level of grid interoperability. We defined the essential requirements of a novel middleware tool called Meta-Broker, proposed a general architecture, and have shown how it can be implemented in two different grid portal environments. We derived two language schemas for describing resource brokers and scheduling policies. This

metadata plays an important role in the scheduling process at the meta-brokering level. Using this meta-information about inter-connected resource brokers and user requests we proposed a general matchmaking algorithm for meta-brokering scheduling policies. We have also envisaged an interoperable Global Meta-Brokering Service as a SOKU that will be able to serve user requests in future generation grids. Our future work will aim at finalizing the presented meta-brokering solutions and carrying out performance measurements in the two portal environments. Besides, we will investigate the requirements of the meta-brokering SOKU, and design scheduling policies for peer-2-peer meta-brokering. Once we achieve these goals and other grid middleware services will also be mature enough, we can inter-connect the meta-brokering instances to bring the meta-brokering SOKU to life and establish the World Wide Grid.

5 Acknowledgement

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265), within the framework of its Researcher Exchange Programme no. 17.

References

[1] Next Generation Grids Report: Future for European Grids: GRIDs and Service Oriented Knowledge Utilities – Vision and Research Directions 2010 and Beyond, December 2006 (NGG3)

[2] K. Krauter, R. Buyya and M. Maheswaran, A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, Software: Practice and Experience (SPE), ISSN: 0038-0644, Volume 32, Issue 2, Pages: 135-164, Wiley Press, USA, February 2002.

[3] A. Kertesz, P. Kacsuk, A Taxonomy of Grid Resource Brokers, 6th Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2006) in conjunction with the Austrian Grid Symposium 2006, Innsbruck, pp. 201-210, Austria, September 21-23, 2006

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

[6] 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, 2006). CCGRID. IEEE Computer Society, Washington, DC, 52.

[7] 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

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

[9] 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.

[10] A. Iosup, D. H.J. Epema, T. Tannenbaum, M. Farrellee, M. Livny, Inter-Operating Grids through Delegated MatchMaking, in proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC07), Reno, Nevada, November 2007.

[11] T. Vazquez, E. Huedo, R. S. Montero, I. M. Llorente, Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures, pp. 372-381, Euro-Par 2007, August 28, 2007

[12] A. Kertesz, 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.

[13] I. Rodero, J. Corbalan, F. Guim, L. L. Fong, Y. G. Liu, and S. Masoud Sadjadi, Looking for an evolution of grid scheduling: Meta-brokering, In Proceedings of the Second CoreGRID Workshop on Middleware at ISC2007, Dresden, Germany, June 2007.

[14] P. Kacsuk, G. Sipos, Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal, Journal of Grid Computing, Feb 2006, pp. 1-18.

[15] I. Rodero, F. Guim, J. Corbalan, J. Labarta, eNANOS: Coordinated Scheduling in Grid Environments, Parallel Computing 2005, pp. 81-88, Malaga, Spain, 12-16 September, 2005.

[16] A. Kertesz, P. Kacsuk, Meta-Broker for Future Generation Grids: A new approach for a high-level interoperable resource management, CoreGRID Workshop on Grid Middleware in conjunction with ISC'07 conference, Dresden, Germany, June 25-26, 2007.

[17] A. Kertesz, I. Rodero and F. Guim, Data Model for Describing Grid Resource Broker Capabilities, CoreGRID Workshop on Grid Middleware in conjunction with ISC'07 conference, Dresden, Germany, June 25-26, 2007.

6 Appendix: XML schemas of BPDL 2.0 and MBSDL

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <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">
3    <xsd:import namespace="uri:MBSchedulingDescriptionLanguage" schemaLocation="mbsdl.xsd"/>
4    <xsd:element name="BPDL" type="bpdl:BPDL_Type">
5      <xsd:annotation>
6        <xsd:documentation>Broker Property Description Language 2.0</xsd:documentation>
7      </xsd:annotation>
8    </xsd:element>
9    <xsd:complexType name="BPDL_Type">
10     <xsd:sequence>
11       <xsd:any namespace="##other" processContents="lax"/>
12       <xsd:element name="BrokerID" type="bpdl:BrokerID_Type"/>
13       <xsd:element name="Interface" type="bpdl:InterfaceType" maxOccurs="unbounded"/>
14       <xsd:element name="Monitoring" type="bpdl:Monitoring_Type"/>
15       <xsd:element name="Security" type="bpdl:Security_Type"/>
16       <xsd:element name="PerformanceMetrics" type="bpdl:PerformanceMetrics_Type"/>
17       <xsd:element ref="mbsdl:SDL"/>
18     </xsd:sequence>
19     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
20     <xsd:attribute name="description" type="xsd:string" use="optional"/>
21     <xsd:attribute name="targetNameSpace" type="xsd:anyURI" use="optional"/>
22     <xsd:anyAttribute namespace="##other" processContents="lax"/>
23   </xsd:complexType>
24   <xsd:complexType name="BrokerID_Type">
25     <xsd:simpleContent>
26       <xsd:extension base="xsd:string">
27         <xsd:anyAttribute namespace="##other" processContents="lax"/>
28       </xsd:extension>
29     </xsd:simpleContent>
30   </xsd:complexType>
31   <xsd:complexType name="InterfaceType">
32     <xsd:sequence>
33       <xsd:element name="type" type="bpdl:InterfacesEnumeration"/>
34       <xsd:element name="name" type="xsd:string"/>
35       <xsd:element name="description" type="xsd:string" minOccurs="0"/>
36       <xsd:element name="Parameters" minOccurs="0">
37         <xsd:complexType>
38           <xsd:sequence>
39             <xsd:element name="Parameter" maxOccurs="unbounded">
40               <xsd:complexType>
41                 <xsd:sequence>
42                   <xsd:element name="description" minOccurs="0"/>
43                 </xsd:sequence>
44                 <xsd:attribute name="name" type="xsd:NCName"/>
45                 <xsd:attribute name="type" type="xsd:NCName"/>
46               </xsd:complexType>
47             </xsd:element>
48           </xsd:sequence>
49         </xsd:complexType>
50       </xsd:element>
51       <xsd:element name="ReturnedValue" minOccurs="0">
52         <xsd:complexType>
53           <xsd:sequence>
54             <xsd:element name="description" minOccurs="0"/>
55           </xsd:sequence>
56           <xsd:attribute name="name" type="xsd:NCName"/>
57           <xsd:attribute name="type" type="xsd:NCName"/>
58         </xsd:complexType>
59       </xsd:element>
60     </xsd:sequence>
61   </xsd:complexType>
```

```
62  <xsd:complexType name="MonitoringMetric_Type">
63    <xsd:sequence>
64      <xsd:element name="Name" type="xsd:string"/>
65      <xsd:element name="Description" type="xsd:string"/>
66    </xsd:sequence>
67    <xsd:attribute name="MetricType" type="bpdl:MonitoringInfoEnumeration" use="required"/>
68    <xsd:anyAttribute namespace="##other"/>
69  </xsd:complexType>
70  <xsd:complexType name="Monitoring_Type">
71    <xsd:sequence>
72      <xsd:element name="Metric" type="bpdl:MonitoringMetric_Type" maxOccurs="unbounded"/>
73    </xsd:sequence>
74    <xsd:attribute name="accessMethod" type="xsd:string" use="optional"/>
75    <xsd:anyAttribute namespace="##other"/>
76  </xsd:complexType>
77  <xsd:complexType name="Security_Type">
78    <xsd:choice>
79      <xsd:element name="MyProxy" type="bpdl:MyProxy_Type"/>
80      <xsd:element name="OtherSecurity" type="bpdl:OtherSecurity_Type"/>
81    </xsd:choice>
82    <xsd:anyAttribute namespace="##other"/>
83  </xsd:complexType>
84  <xsd:complexType name="MyProxy_Type">
85    <xsd:sequence>
86      <xsd:element name="IsSupported" type="xsd:boolean"/>
87      <xsd:element name="ServerName" type="xsd:string" minOccurs="0"/>
88      <xsd:element name="PortNumber" type="xsd:int" minOccurs="0"/>
89    </xsd:sequence>
90    <xsd:anyAttribute namespace="##other"/>
91  </xsd:complexType>
92  <xsd:complexType name="OtherSecurity_Type">
93    <xsd:sequence>
94      <xsd:element name="Details" type="xsd:string"/>
95    </xsd:sequence>
96    <xsd:attribute name="name" type="xsd:NCName"/>
97    <xsd:anyAttribute namespace="##other"/>
98  </xsd:complexType>
99  <xsd:complexType name="PerformanceMetrics_Type">
100   <xsd:sequence>
101     <xsd:element name="AVGWaitingTime" type="bpdl:PerformanceMetric_Type"/>
102     <xsd:element name="AVGSlowdown" type="bpdl:PerformanceMetric_Type"/>
103     <xsd:element name="FinishedJobs" type="bpdl:PerformanceMetric_Type"/>
104     <xsd:element name="FailedJobs" type="bpdl:PerformanceMetric_Type"/>
105     <xsd:element name="OtherMetric" type="bpdl:PerformanceMetric_Type"
            maxOccurs="unbounded"/>
106     <xsd:element name="Prediction" type="bpdl:PerformanceMetric_Type" minOccurs="0"
            maxOccurs="unbounded"/>
107   </xsd:sequence>
108   <xsd:anyAttribute namespace="##other"/>
109 </xsd:complexType>
110 <xsd:complexType name="PerformanceMetric_Type">
111   <xsd:sequence>
112     <xsd:element name="name" type="xsd:string"/>
113     <xsd:element name="description" type="xsd:string"/>
114     <xsd:element name="value" type="xsd:string"/>
115   </xsd:sequence>
116   <xsd:anyAttribute namespace="##other"/>
117 </xsd:complexType>
118 <xsd:simpleType name="InterfacesEnumeration">
119   <xsd:restriction base="xsd:string">
120     <xsd:enumeration value="Submit"/>
121     <xsd:enumeration value="Cancel"/>
122     <xsd:enumeration value="Suspend"/>
123     <xsd:enumeration value="Resume"/>
124     <xsd:enumeration value="Migrate"/>
125     <xsd:enumeration value="other"/>
126   </xsd:restriction>
127 </xsd:simpleType>
```

```
128     <xsd:simpleType name="MonitoringInfoEnumeration">
129       <xsd:restriction base="xsd:string">
130         <xsd:enumeration value="StaticInfo"/>
131         <xsd:enumeration value="DynamicInfo"/>
132         <xsd:enumeration value="AggregatedInfo"/>
133         <xsd:enumeration value="other"/>
134       </xsd:restriction>
135     </xsd:simpleType>
136   </xsd:schema>
137
138
139   <?xml version="1.0" encoding="UTF-8"?>
140   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:mbsdl="uri:MBSchedulingDescriptionLanguage"
        targetNamespace="uri:MBSchedulingDescriptionLanguage" elementFormDefault="qualified"
        attributeFormDefault="unqualified">
141     <xsd:element name="SDL" type="mbsdl:SDL_Type">
142       <xsd:annotation>
143         <xsd:documentation>MB Scheduling Description Language</xsd:documentation>
144       </xsd:annotation>
145     </xsd:element>
146     <xsd:complexType name="SDL_Type">
147       <xsd:sequence>
148         <xsd:any namespace="##other" processContents="lax"/>
149         <xsd:element name="Constraints" type="mbsdl:Constraints_Type"/>
150         <xsd:element name="QoS" type="mbsdl:QoS_Type"/>
151         <xsd:element name="Policy" type="mbsdl:Policy_Type"/>
152       </xsd:sequence>
153       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
154       <xsd:attribute name="description" type="xsd:string" use="optional"/>
155       <xsd:attribute name="targetNameSpace" type="xsd:anyURI" use="optional"/>
156       <xsd:anyAttribute namespace="##other" processContents="lax"/>
157     </xsd:complexType>
158     <xsd:complexType name="Constraints_Type">
159       <xsd:sequence>
160         <xsd:any namespace="##other"/>
161         <xsd:element name="Middleware" type="mbsdl:Middleware_Type" maxOccurs="unbounded"/>
162         <xsd:element name="JobType" type="mbsdl:JobTypeEnumeration" maxOccurs="unbounded"/>
163         <xsd:element name="Time" type="mbsdl:Time_Type"/>
164         <xsd:element name="Budget" type="xsd:long"/>
165         <xsd:element name="RemoteFileAccess" type="mbsdl:RemoteFileAccessEnumeration"
              minOccurs="0" maxOccurs="unbounded"/>
166         <xsd:element name="OtherConstraint" type="mbsdl:Other_Type" maxOccurs="unbounded"/>
167       </xsd:sequence>
168     </xsd:complexType>
169     <xsd:complexType name="Middleware_Type">
170       <xsd:sequence>
171         <xsd:element name="GridName" type="mbsdl:GridNameEnumeration" minOccurs="0"/>
172         <xsd:element name="ProxyName" type="xsd:string" minOccurs="0"/>
173         <xsd:element name="MYProxy" type="mbsdl:MyProxy_Type" minOccurs="0"/>
174         <xsd:element name="VirtualOrganisation" type="mbsdl:VirtualOrganisation_Type"
              minOccurs="0" maxOccurs="unbounded"/>
175         <xsd:element name="InformationSystem" type="mbsdl:InformationSystem_Type"
              minOccurs="0"/>
176         <xsd:any namespace="##other" minOccurs="0"/>
177       </xsd:sequence>
178       <xsd:anyAttribute namespace="##other" processContents="lax"/>
179     </xsd:complexType>
180     <xsd:complexType name="VirtualOrganisation_Type">
181       <xsd:sequence>
182         <xsd:element name="InformationSystem" type="mbsdl:InformationSystem_Type"/>
183         <xsd:element name="ProxyName" type="xsd:string" minOccurs="0"/>
184         <xsd:any namespace="##other" minOccurs="0"/>
185       </xsd:sequence>
186       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
187       <xsd:anyAttribute namespace="##other" processContents="lax"/>
188     </xsd:complexType>
189     <xsd:complexType name="InformationSystem_Type">
```

```
190       <xsd:sequence>
191         <xsd:element name="MDS" type="xsd:string" minOccurs="0"/>
192         <xsd:element name="BDII" type="xsd:string" minOccurs="0"/>
193         <xsd:element name="WebMDS" type="xsd:string" minOccurs="0"/>
194         <xsd:any namespace="##other" minOccurs="0"/>
195       </xsd:sequence>
196       <xsd:attribute name="name" type="xsd:NCName" use="required"/>
197       <xsd:anyAttribute namespace="##other" processContents="lax"/>
198     </xsd:complexType>
199     <xsd:complexType name="QoS_Type">
200       <xsd:sequence>
201         <xsd:any namespace="##other"/>
202         <xsd:element name="Agreement" type="mbsdl:Agreement_Type" minOccurs="0"
              maxOccurs="unbounded"/>
203         <xsd:element name="FaultTorelanceMechanisms" type="mbsdl:FaultToleranceEnumeration"
              maxOccurs="unbounded"/>
204         <xsd:element name="AdvanceReservation" type="mbsdl:AdvanceReservation_Type"
              minOccurs="0"/>
205         <xsd:element name="Priority" type="xsd:string" minOccurs="0"/>
206         <xsd:element name="GridAccessControl" type="xsd:string" minOccurs="0"/>
207         <xsd:element name="EmailNotification" type="xsd:string" minOccurs="0"/>
208       </xsd:sequence>
209       <xsd:anyAttribute namespace="##other" processContents="lax"/>
210     </xsd:complexType>
211     <xsd:complexType name="BrokerName_Type">
212       <xsd:simpleContent>
213         <xsd:extension base="xsd:string">
214           <xsd:anyAttribute namespace="##other" processContents="lax"/>
215         </xsd:extension>
216       </xsd:simpleContent>
217     </xsd:complexType>
218     <xsd:complexType name="Policy_Type">
219       <xsd:sequence>
220         <xsd:element name="PolicyName" type="mbsdl:PolicyEnumeration" minOccurs="0"/>
221         <xsd:element name="OtherPolicy" type="mbsdl:Other_Type" minOccurs="0"/>
222         <xsd:element name="LRMSPolicy" type="mbsdl:Other_Type" minOccurs="0"/>
223       </xsd:sequence>
224       <xsd:anyAttribute namespace="##other"/>
225     </xsd:complexType>
226     <xsd:complexType name="Time_Type">
227       <xsd:sequence>
228         <xsd:element name="StartTime" type="xsd:date"/>
229         <xsd:element name="Duration" type="xsd:long"/>
230         <xsd:element name="TimeOut" type="xsd:long"/>
231       </xsd:sequence>
232       <xsd:anyAttribute namespace="##other"/>
233     </xsd:complexType>
234     <xsd:complexType name="Other_Type">
235       <xsd:sequence>
236         <xsd:element name="Name" type="xsd:string"/>
237         <xsd:element name="Value" type="xsd:string"/>
238       </xsd:sequence>
239       <xsd:anyAttribute namespace="##other"/>
240     </xsd:complexType>
241     <xsd:complexType name="MyProxy_Type">
242       <xsd:sequence>
243         <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
244         <xsd:element name="ServerName" type="xsd:string"/>
245         <xsd:element name="PortNumber" type="xsd:int" minOccurs="0"/>
246       </xsd:sequence>
247       <xsd:anyAttribute namespace="##other"/>
248     </xsd:complexType>
249     <xsd:complexType name="Agreement_Type">
250       <xsd:sequence>
251         <xsd:element name="Target" type="xsd:anyURI"/>
252         <xsd:element name="ConfidenceLevel" type="xsd:positiveInteger"/>
253       </xsd:sequence>
254       <xsd:anyAttribute namespace="##other"/>
```

```
255     </xsd:complexType>
256     <xsd:complexType name="AdvanceReservation_Type">
257       <xsd:sequence>
258         <xsd:element name="ResourceName" type="xsd:string"/>
259         <xsd:element name="Date" type="xsd:date"/>
260       </xsd:sequence>
261       <xsd:anyAttribute namespace="##other"/>
262     </xsd:complexType>
263     <xsd:simpleType name="GridNameEnumeration">
264       <xsd:restriction base="xsd:string">
265         <xsd:enumeration value="GT2"/>
266         <xsd:enumeration value="GT3"/>
267         <xsd:enumeration value="GT4"/>
268         <xsd:enumeration value="EGEE-LCG-2"/>
269         <xsd:enumeration value="EGEE-gLite"/>
270         <xsd:enumeration value="NorduGrid"/>
271         <xsd:enumeration value="Unicore"/>
272       </xsd:restriction>
273     </xsd:simpleType>
274     <xsd:simpleType name="JobTypeEnumeration">
275       <xsd:restriction base="xsd:string">
276         <xsd:enumeration value="Serial"/>
277         <xsd:enumeration value="Mpi"/>
278         <xsd:enumeration value="Pvm"/>
279         <xsd:enumeration value="Checkpointable"/>
280         <xsd:enumeration value="Interactive"/>
281         <xsd:enumeration value="Threads"/>
282         <xsd:enumeration value="OpenMP"/>
283         <xsd:enumeration value="Mpi+OpenMP"/>
284         <xsd:enumeration value="Caf"/>
285         <xsd:enumeration value="Upc"/>
286       </xsd:restriction>
287     </xsd:simpleType>
288     <xsd:simpleType name="RemoteFileAccessEnumeration">
289       <xsd:restriction base="xsd:string">
290         <xsd:enumeration value="GridFTP"/>
291         <xsd:enumeration value="RFT"/>
292         <xsd:enumeration value="GASS"/>
293         <xsd:enumeration value="Unicore"/>
294         <xsd:enumeration value="SRB"/>
295         <xsd:enumeration value="EGEE-LFN"/>
296       </xsd:restriction>
297     </xsd:simpleType>
298     <xsd:simpleType name="PolicyEnumeration">
299       <xsd:restriction base="xsd:string">
300         <xsd:enumeration value="ScheduleByCpu"/>
301         <xsd:enumeration value="ScheduleByMemory"/>
302         <xsd:enumeration value="ScheduleByDiskSize"/>
303         <xsd:enumeration value="RandomHost"/>
304       </xsd:restriction>
305     </xsd:simpleType>
306     <xsd:simpleType name="FaultToleranceEnumeration">
307       <xsd:restriction base="xsd:string">
308         <xsd:enumeration value="Checkpointing"/>
309         <xsd:enumeration value="Rescheduling"/>
310         <xsd:enumeration value="Replication"/>
311       </xsd:restriction>
312     </xsd:simpleType>
313   </xsd:schema>
```