

# Using Semantic Technologies for Resource Allocation in Computing Service Providers

Jorge Ejarque, Marc de Palol, Iñigo Goiri, Ferran Julia, Jordi Guitart, Jordi Torres and Rosa M. Badia  
Barcelona Supercomputing Center and Universitat Politècnica de Catalunya  
Jordi Girona 31, 08034 Barcelona, Spain  
{jorge.ejarque, marc.de.palol, inigo.goiri, ferran.julia, jordi.guitart, jordi.torres, rosa.m.badia}@bsc.es

## Abstract

*Service providers business goals require an efficient management of their computational resources in order to perform provisioning, deployment, execution and adaptation. While traditionally those activities require human intervention, the current trend is to automatize those processes in order to reduce costs and increase productivity. The Semantically-Enhanced Resource Allocator is a framework designed to obtain an efficient autonomous service provider management while fulfilling customers requests. While the efficient dynamic resource management is obtained by means of virtualization in this paper we focus on the task scheduling and resource allocation tasks. The task scheduling and resource allocation processes are enhanced by semantically describing the tasks and the resources and using these descriptions to infer the resource assignments. A resource ontology has been extended to take into account all the features considered in our system, and as a proof of concept, a small set of rules that define the behavior of the inference process have been defined. The proposed framework has been implemented in the scope of the BREIN European project, which aims to develop an infrastructure to enable service providers to reduce costs whilst maximizing profit by using technologies such as agents and semantics.*

## 1 Introduction

In the framework of the Service-Oriented Computing (SOC) paradigm [10], services can be easily composed to build distributed applications. Those services are offered by Service Providers (SP) which provide the implementation, the service description and technical and business support. From the business point of view, the SP agrees with its customers the Quality of Service (QoS) and level of service through a Service Level Agreement (SLA). Generally, the SLA is a bilateral contract between the customer and the

SP that states not only the conditions of service, but also the penalties when the service is not satisfied and metrics to evaluate the level of service.

Project BREIN (Business objective driven REliable and Intelligent grids for real busiNess) [1] has as objective bringing the concepts developed in Grid research projects, namely the concept of so-called dynamic virtual organizations towards a more business-centric model, by enhancing the system with methods from artificial intelligence, intelligent systems, semantic web, etc. In the scenario considered in BREIN, a customer submits a goal to be achieved. This goal is translated into an abstract workflow and for each step a business process will be started in order to find out the external suppliers of the services that could perform each step.

The aim of our work deals with the case of a SP providing computational services in such an scenario. The assumption is that the SP is able to provide different type of computation services, i.e. transactional applications and long running batch jobs. This will imply not only an heterogeneous workload but also different type of SLAs agreed with the different customers. We consider then that the SP would like to meet the following objectives:

- Maximize profit, by making an optimal use of the resources and reducing the penalties incurred by not meeting the SLA terms
- Given that the SP may have an level of preference (maybe subjective, driven by business goals) of the customers, maximize the satisfaction of the customers, and in case of conflict, penalize first those customers with less priority.

Under these premises, a Semantically-Enhanced Resource Allocator has been designed and implemented that benefits from technologies such as semantics, agents and virtualization. The objective is to provide an environment that is able to statically schedule the customer requests taking into account not only the SLA terms, but also the state of the SP resources and the SP level of preference for each

customer. Besides, we want to provide the system with the ability of re-schedule requests to meet the requirements of more priority ones and of consider advanced reservations. With this objective, the whole environment is semantically described and the scheduler is able to infer the resource assignments based on a set of rules. Additionally, virtualization is used as a mean for providing a full customized and isolated virtual environment for each application but at the SP level, the system supports fine-grain dynamic resource distribution among these virtual environments in order to adapt to changing resource requirements of the applications and maximizing SP goals.

The structure of the paper is the following: section 2 summarizes the related work that can be found in the literature, section 3 describes the architecture of the Semantically-Enhanced Resource Allocator, section 4 presents the resource ontology used to describe our environment. In section 5 we describe how the semantic description annotation is automatically performed in the system. Section 6 presents the semantic scheduler, which is the engine that performs the scheduling decisions based on the semantically enriched metadata available and in a set of inference rules. Section 7 presents some experimental results obtained with the implementation of the system. Finally, section 8 concludes the paper.

## 2 Related Work

Recently, there has been a significant interest of enriching Grid resources and services with semantic descriptions to make them machine understandable[12]. Traditionally, each organization in grids published their resource properties and application requirements using their own language and meanings making interoperability more expensive. The Semantic Grid [12] tackles this problem introducing semantic web technologies in Grids.

The EU FP6 OntoGrid project [8] proposes a reference model to support semantics in Grids called Semantic OGSA [5] which consists of a set of OGSA compliant services in charge of managing semantic descriptions bound to a grid resource, ontologies, rules and reasoning engines. In resource allocation problem [15] and [17] present two different ontology-based resource matchmaking algorithms implemented by a set of rules which identify the resources which fulfill the requirements, and in the Phosphorus project [11], semantics are used to automatically select resources. However, the resource matchmaking is only a part of the resource allocation and job scheduling. GraniteNights [6] is a multi-agent system that uses semantic descriptions to organize evening plans under user constraints. Although the system performs similar resource selection and scheduling features to those solved by our proposal (abstracting the problem), the scheduling done in their case is

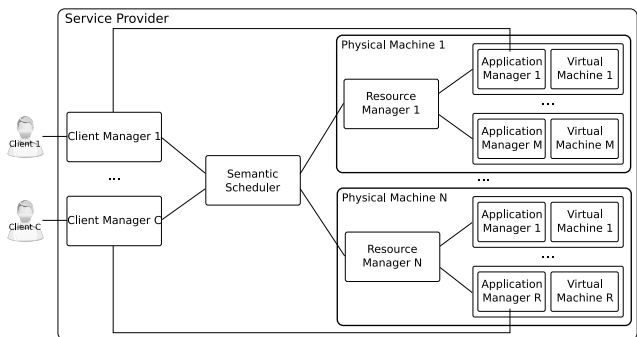
static, without re-scheduling and without considering neither resource usage nor SLA establishment.

In the approach presented in this paper, we have used semantics in the whole resource allocation process, that is, resource matchmaking and allocation of tasks in the selected resources. We have also taken into account not only the task requirements, but also business parameters such as the customer priority. Finally, we also want to mention other interesting previous work about semantic support for scheduling and ontologies for resource access coordination have been presented in [9] and [13]. We have partially used their results to build our resource allocator.

## 3 Overall Architecture of the Semantically-Enhanced Resource Allocator

This section presents the main components in our proposal, giving an overview of the global architecture and describing the interactions among these components. Figure 1 shows the main four components in the system (Client Manager, Semantic Scheduler, Resource Manager and Application Manager) and how they interact with each other and the Semantic Metadata Repository, which manages metadata related to each system entity.

Each component contains an agent and a core component in order to improve system autonomy and take profit of agents abstraction. The agent is in charge of the communication between components, task execution monitoring, detecting undesired events (SLA violations, failures,...) and coordinating the reaction of a component to these events. The agent wraps the core component functionalities by means of a set of agent behaviors which basically call methods of the core components. The reason of this wrapping is because the inter-component communication is easier with agents, and some of the required features are provided by a specific behavior also give more autonomy to the system as they are reactive components.



**Figure 1. Semantically-enhanced Resource Allocator architecture**

Following we describe the basic functionality of the mentioned components. The Client Manager (CM) manages the client's task execution by requesting the required resources and running jobs. In addition, it makes decisions about what must to be done when unexpected events such as SLA violations happen.

The Semantic Scheduler allocates resources to each client's task taking into account its requirements, its priority and the status of the system, in such a way that the clients with more priority are favored. Allocation decisions are derived using an inference engine using the semantic descriptions of jobs and physical available resources. This descriptions are made by the users (for the jobs) and by the system administrators (for the physical resources). This descriptions are made according to the system Ontology, described with the RDF language and stored in the Semantic Metadata Repository.

The Resource Manager (RM) creates a virtual machine (VM) to execute each client's task according to the minimum resource allocation (CPU, memory, disk space...) given by the Scheduler and the task requirements (e.g required software). Once the VM is already created, it dynamically redistributes the remaining resources among the different client's tasks depending on the resource usage of each running task, its priority and the SLA status (i.e. is it being violated?). The latter information is provided by the Application Manager. This resource redistribution mechanism allows the system incrementing the allocated resources to a task that needs them by reducing the assignment to other tasks that are not using them.

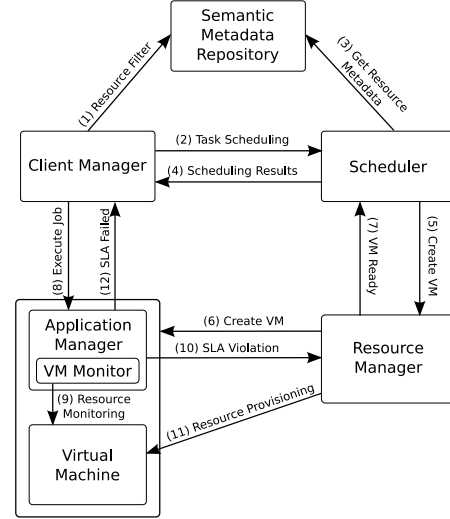
Finally, the Application Manager (AM) monitors the resource usage and the SLA parameters in order to evaluate if a SLA it being violated. A SLA violation can be solved by requesting more resources to the RM. If the RM cannot provide more resources, the AM will forward the request to the Client Manager.

### 3.1 Task lifecycle

Figure 2 shows the task lifecycle in the overall system and the information that is sent among the different components. Initially, at boot time, every component gets its semantic description and stores it in the Semantic Metadata Repository. An interaction starts when a task arrives to the system and a Client Manager (CM) is created in order to manage its execution. The CM registers the task in the Semantic Metadata Repository and preselects potential nodes for running the task by making a query with inference to the repository (1). Then it sends a time slot request to the Scheduler specifying the required resources for the task and the list of potential nodes (2).

In this stage, the Scheduler uses the metadata of the system stored in the Semantic Metadata Repository (3) to infer

in which node will the task be executed. At this point the Scheduler informs the CM whether the task has been successfully scheduled or canceled (4). When the time to execute the task arrives, the Scheduler contacts with the RM in charge of the node where the task has been allocated and asks for the creation of a VM for executing the task (5).



**Figure 2. Task lifecycle**

When the RM receives the Scheduler request, it creates a VM and an AM that monitors the SLA fulfillment for this task (6). Once the VM is ready to be used, the Scheduler is informed (7) and it sends a message to the CM indicating the access information to that VM. At this point, the CM can submit the task to the newly created VM (8).

From this moment, the task executes in a VM which is being monitored by the AM in order to detect SLA violations (9). If this occurs, the AM requests for more resources to the RM (10), trying to solve the SLA violation locally to the node (11). This request for more resources is done as many times as needed until the SLA is not violated any more or the RM tells to the AM that the requested resource increment is not possible. In the latter case, since solving the SLA violation locally is not possible, the AM informs the CM about this situation (12). In this case, the CM can decide to resubmit the task with higher resource requirements or notify the client if the resubmission fails.

This papers focuses mainly in Semantic Scheduler and how the semantics are managed in the system. Additional description of the other components can be found in [7].

## 4 A Resource Allocation Ontology

The ontology presented in this section try to describe the entities related to the resource allocation process within a

service provider, where the different resources are assigned to the tasks submitted to the system depending on their requirements and the priority of the clients who have submitted the task. This resource allocation ontology is based on the work presented in [13]. It describes *Agents* (as *Requesters* or *Providers*), *Activities* (a.k.a tasks), *Resources* and relations between them in order to describe how the usage of provider's resources by the requester's tasks can be coordinated. Despite it was designed for resource usage coordination, it presents many similarities with resource allocation process and some classes which can be shared. The Client Agent and Resource Manager components can be mapped as subclasses of original *Requester* and *Provider Agents* and finally the different client *Tasks* can be mapped as a kind of the *AtomicActivity* class which describes an *Activity* which can not be decompose in other ones. The *Activity* class describe a set of useful properties for the task scheduling such as the task state (i.e proposed, running, done, etc), the earliest start and deadline times as well as the task duration.

The coordination ontology also define a set of *Interdependencies* between two *AtomicActivities* which show us if both tasks can be executed in parallel or one of them must wait until the end of the other one. Finally, it also provides us a general *Resource* class which defines a set of core resource properties. The most important ones for our purpose are: *sharable* property which means if a resource can be shared by several resources; the *consumable* property which shows if the usage of this resource can make it unavailable; and the *clonable* property which describe if the resource can be copied. Although the original ontology describe a large part of our system, it is still uncompleted for covering the requirements of the resource allocation process performed by our system. Therefore, some changes and extensions have been required in order to make the coordination ontology suitable for resource allocation. These changes are explained in the following paragraphs.

The main classes of the resource allocation ontology are depicted in figure 3. As we can see on that figure, the *Resource* class has been extended with different subclasses covering the main resources used by a Service Provider. There are some existent models for computing resource description such as GLUE [4] schema used by the Globus or the Unicore[16] schema or the Common Information Model[2] between others. For this first implementation, we do not intend to cover a complete description of the resources, we only define a set of resource types and properties which are common in the mentioned models and are enough to validate our semantic resource allocation. We have define a *HardwareResource*, as sharable, consumable and not cloneable resource which includes *CPU*, *Memory*, *Disk* and *Network*. Another type of resources are the *SoftwareResource* which is defined as sharable and is grouped

in *Images*, and the *File* which describe shared files required by different *Tasks*. The *File* resource is defined as shareable and cloneable and it is stored in *Disks*. Besides these definitions, we require another type of resource due to the fact that *Tasks* can not be directly executed on the resource, but by means of *Hosts* which contain a set of *Hardware* and *Software* resources.

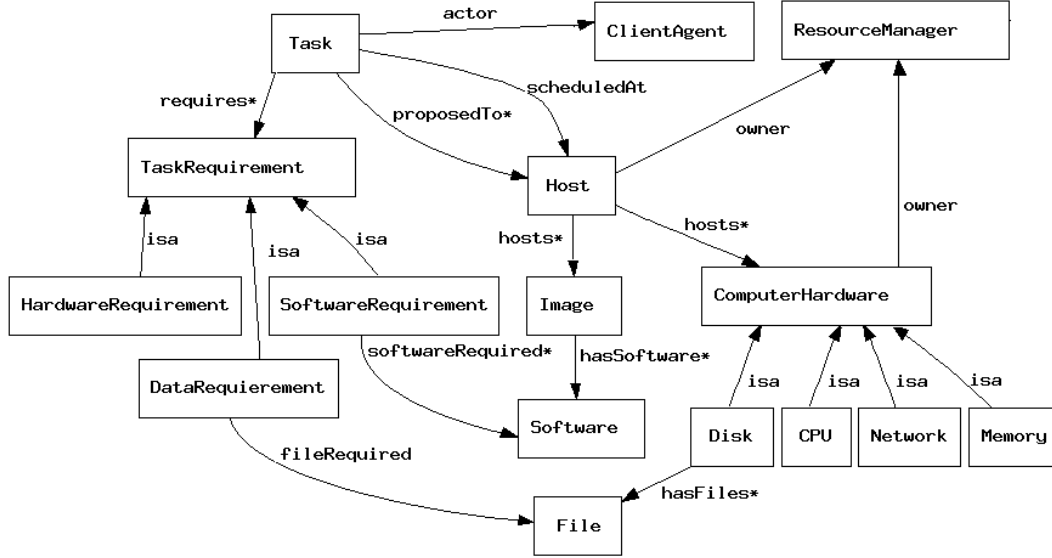
Another important change is the modification of the *Task* description. The original *AtomicActivity* was thought to require a single instance of *Resource*. However, this single resource instance does not match with the resource allocation for two reasons. First, because a *Task* could require more than one resource, and second, because this resource is unknown. The resource allocator try to find the best resources for each client task. In order to cover this difference, the *requires* property has been changed. Instead of containing a single resource instance it will contain a set of *TaskRequirements*. This new class describes the abstract resources required by a client task. These requirements are classified in subclasses according to the type of resource required. These subclasses include:

- *HardwareRequirements* such as required CPU, amount of memory, network bandwidth.
- *SoftwareRequirements* such as the required Operating system, libraries or software.
- *DataRequirements* such as the required input files and the output files.

Hardware and software requirements are used in the match-making process to look for hosts which fulfil those requirements. On the other hand, Data requirements can be used for other issues, such as to detect data dependencies and exploit data locality, considering the host that already has the required files accessible in order to avoid unnecessary data transfers.

In addition to the changes performed in the *requires* property, new properties are added to the *Task* description. One of them is the *proposedHosts* property which contains a list of *Hosts* which fulfil the task requirements, and the other one is the *scheduledAt* property which contains the assigned hosts where the task is going to be executed.

Finally, the *ClientAgent* (a *Provider Agent*) includes the business parameters. For this initial prototype we have only included the *priority* property which shows how important is this client for the Service Provider, however, it will be easily extended with other bussines parameters such as rewards, penalties, non-payments, etc. This resource allocation ontology is used by the Client and the Resource Manager to describe the client tasks and the Service Provider's resources and by the Semantic Scheduler to infer a decision about which are the best machines and time slot to schedule each client task, as it is explained in the following sections.



**Figure 3. Ontology for Resource Allocation**

## 5 Automatic Annotation of Semantic Descriptions

The most important part using semantics in the resource allocation and job scheduling process is annotation of the semantic description of all the entities involved on it. This annotation includes the search of the relevant metadata to be included into the semantic description, create the semantic models and register them into a semantic metadata repository to make these descriptions available for all the components that will require them. The usage of semantically enriched metadata should not require additional data management of the Clients or System Administrators, so it should be performed automatically and transparent to the user. In the Semantically-Enhanced Resource Allocator, this task is performed by the Client Manager and the Resource Manager.

### 5.1 Getting the resource description: Resource Manager

As it has been described in previous sections, each host of our system is managed by a Resource Manager. It is started at the machine boot time and deploys an agent whose first task is to collect all the information required to fill the semantic description. It is obtained by parsing the data published by a resource monitor such as Ganglia [3]. Then, an RDF model is created with the collected data using the *Host* and *HardwareResources* classes of the resource allocation ontology. In addition to the resource properties, the Resource Manager description is added to the RDF, includ-

ing the address to contact with its Agent. Once, the RDF model is created, it is registered in the Semantic Metadata Repository where all metadata will be stored and will be accessible for the Client Manager and the Semantic Scheduler.

### 5.2 Getting the task descriptions: Client Manager

When a new task submission request enters the system, a CM Agent is created to manage the task execution in the system. It gets the task description provided by the client and creates an RDF model including *TaskRequirements* and the client who request the task. This task model is created according to our ontology and registered into the Semantic Framework. Simultaneously to reading the task description and creating the RDF model, the client agents is using the task requirements to build a semantic query for RDF metadata described in SPARQL[14]. This query is used to select from the Semantic Metadata Repository all the hosts that fulfil the software and hardware requirements of the task. The query results are also inserted in the task model and registered. Performing this step in the CM, we unload the Scheduler of the task-machine matchmaking and it is also a first step toward a fully distributed scheduling system where the Client Manager negotiates with the selected resource managers.

Regarding the Client information, we assume that a contract has been agreed by client and service provider and that due to the historical client data the Service provider administrator will be assigned a priority for this client, and that all the client data have been previously registered in the Se-

semantic Metadata Repository.

## 6 Semantic Scheduler

The Semantic Scheduler is a proof of concept implementation of decision taking using reasoning, rules with complementary builtins and agents. The reason for the use of semantic technologies in the scheduling process is to explore new alternatives to perform more complex tasks' schedulings taking into account technical and business parameters, as well as, to exploit the flexibility and extensibility provided by the Semantic technologies in terms of compatibility with other systems and easiness of extending or changing scheduling policies.

The overview of the Semantic scheduler is that a rule engine infers a task scheduling into the available resources from the metadata described semantically evaluating a set of rules. Once the Scheduler receives a scheduling request from the CM, it performs the following tasks:

### 6.1 Get the required semantically enriched metadata

The first step to perform is to obtain semantic description of the data involved in the scheduling process. All this data is maintained united and updated in the Semantic Metadata Repository following the ontology described in section [?] . So all necessary data for the scheduling process is fetched from this repository with a RDF format to be used in the inference. This required metadata is:

- **Semantic Data describing each host:** The process needs data describing the different machines that will take part in the scheduling process. Not just the proposed machines in the task description, also the proposed machines in all scheduled tasks in the system, because this process can reschedule a task into another machine.
- **Semantic Data describing already scheduled/running tasks:** The inference must know all tasks that are in the system, including the ones that are queued and the ones that are executing, also must know their priorities, that is, their owner.
- **Semantic Data describing the clients that are executing tasks in the system:** As stated in the previous bullet, the clients hold, among other metadata the priority, taken as a business data.

### 6.2 Inference

Once the Scheduler has retrieved the data, a Jena Model is created to prepare this data for the inference, attaching the

Resource allocation Ontology and a rule engine to the retrieved data. Once the model is created, the rules to perform the inference must be loaded to the rule engine attached to the model. We have implemented three Jena 2 rules which use a set of auxiliary builtins (rule extensions written as application code) that are fired during the inference process.

- **Rule 1: First Come First Serve.** This rule is fired for all the tasks that are in the system which are in a requested state (recently created). This rule tries to find a time slot into the proposed hosts using the FCFS algorithm, taking into account the hardware requirements and the space left by the other running or scheduled tasks to allocate a new virtual machine.
- **Rule 2: Task reallocation.** If the Rule 1 is not capable to find a time slot and a machine for executing the task, the Rule 2 tries to move scheduled tasks (not already running) from one machine to another one in order to find a time slot for the new task (taking into account hardware requirements and space left as in Rule 1).
- **Rule 3: Less priority task(s) cancellation.** If Rule 2 is not able to find a solution, then the Rule 3 tries to cancel the scheduled tasks (not running) with less priority than the new one.

### 6.3 Collection and interpretation of the results

When the inference process is over, the Jena 2 gives back a deductions' graph. Each deduction in this graph implies a change in the original graph, so it means that there has been some changes in the state of the system, or in other words that a task has been scheduled, re-scheduled or canceled. Then, the scheduler gets all necessary information about the updated tasks evaluating the deductions' graph. After that, all the affected data gets updated in the semantic repository. Task cancellations are notified to their CM and the Scheduler update its scheduled task queue.

At the same time, the Scheduler Agent monitors the execution time of the queued tasks. When the execution time is close, it will contact the RM and CM to start the task execution.

## 7 Experimental Environment and results

We have created a simple testbed to test our prototype. This is a real demonstration we made for testing purposes and to make some preliminary measurement about the overhead introduced in the semantic scheduling.

Our experimental testbed for the whole prototype consists of three machines: the first one is a Pentium D with two CPUs at 3.2GHz with 2GB of RAM that contains the

Semantic Scheduler, the Client Manager and the Semantic Metadata Repository. Two additional machines are used as resource manager. The first resource manager is Pentium with one CPU and 2GB of RAM. The second machine is a 64-bit architecture with 4 Intel Xeon CPUs at 3.0GHz and 6GB of RAM memory. Both Resource Managers, RManager1 and RManager2, can create virtual machines with preinstalled software (such as GT4 or Tomcat) on a debootstrapped Debian Lenny.

Most part of the software is written in Java and runs under a JRE 1.5, except the scripts that manage the creation of the virtual machines, which are written in Bash script, and some libraries used for accessing Xen, which are written in C. As a Semantic Metadata repository we have used Ontokit [8] and Jena 2 as rule engine.

Unifying metrics between machines to define the task requirements and resource capacity is a key issue, specially for processors. For instance, if a machine has two cores at 1,33 GHz it will be different to a quad core at 2GHz each one. We have used a simple approach for quantifying the needed CPU for a given task: the product of desired percentage and the CPU frequency. Due to both hosts has porcessors with a frequency around 3GHz we have normalized the CPU capacity and task requirements to this 3GHz. So the maximum CPU offered by the Resource Managers is shown in table 1.

| Name      | Id  | CPU | Memory |
|-----------|-----|-----|--------|
| RManager1 | RM1 | 400 | 6144   |
| RManager2 | RM2 | 100 | 2048   |

**Table 1. Resources description**

This experiment focuses on the allocation of CPU between Resource Managers. We have designed a test focused in this particular case, nevertheless, these concepts could be applied to other resources like memory, network and bandwidth, etc. In this experiment we send a total of eight tasks to the system. Table 2 describes for each task the requested CPU (*CPU Req.*), the task duration (*Durat.*), deadline *Deadline* and client priority. Deadlines are specified assuming that the use case initial time is 00:00.

The first 4 tasks are sent to test the scheduling when there are enough resources to run the tasks (Rule 1). So, the first task can only be executed on RManager1 because RManager2 can only execute jobs of less than 100% of CPU. It is immediately scheduled and sent to RManager1, which creates the virtual machine to run this task. The 2nd and 3rd task requires also 120%, so it is also scheduled on RManager1. The fourth task, which requires 100% of CPU, is scheduled in RManager2. Tasks demanding more resources will be, then, queued in the Scheduler job queue until the resources are available.

Next three tasks exemplify how the Scheduler resched-

| Id | CPU Req. | Durat. | Deadline | Priority |
|----|----------|--------|----------|----------|
| 1  | 120      | 20'    | -        | Medium   |
| 2  | 120      | 20'    | -        | Medium   |
| 3  | 120      | 20'    | -        | Medium   |
| 4  | 100      | 20'    | -        | Medium   |
| 5  | 100      | 20'    | 00:60    | Low      |
| 6  | 400      | 20'    | 00:40    | Low      |
| 7  | 100      | 20'    | 00:40    | Medium   |
| 8  | 300      | 20'    | 00:60    | High     |

**Table 2. Tasks description**

ules tasks between different Resource Managers. The fifth task is demanding more than the available 40% of CPU, and it is initially scheduled into the RManager1. In this case, the task is not immediately executed, and remains queued in the Scheduler (see Table table 3.a). The sixth task has a shorter deadline that implies that at least has to be scheduled for execution right after the current running tasks (note that we do not interrupt already running jobs). Since this task requires 400% of CPU, it cannot be scheduled into RManager2. But RManager1 cannot accomplish the tasks deadline unless the Scheduler reschedules the fifth task to RManager2 (see Table 3.b). The seventh task has also a sharp deadline, the same reasoning is done, as the deadline cannot be met. The fifth task is again rescheduled after the sixth and the seventh queued in the RManager2 queue (see Table 3.c). The final task will perform a task cancellation. Task 5 and 8 has the same deadline however task 8 cannot be scheduled in RManager 2 because it does not fulfill the requirements. The system cancels task 5 because it the task 5 client has less priority than task 8 one (see Table 3.d). This does not mean that task 5 will not be executed, but that a new scheduling cycle should be started to schedule it.

Additionally, we have simulated the addition of more resources and clients in order to make a small performance analysis of the matchmaking and the semantic scheduler. This simulation is done adding several task and resources descriptions in order to get the expected time in production conditions. We have detected that the most important overhead is the time spent in quering the Semantic Metadata Repository and in the scheduling process. The other overhead created for using semantic technologies such as the semantic registration is done by several components in parallel and it is neglectable compared with the one mentioned above.

Figure 4 shows the time spent on making a SPARQL query vs the number of task. It is obtained adding a new task in to the system. Once the client manager has recieved the task requirements it performs new SPARQL query. As we can see in the figure, the first queries spend much more time to get the results than the other ones. It is due to some initialization issues performed by the Ontokit during the ex-

|           |           |
|-----------|-----------|
| RManager1 | RManager2 |
| Task5     |           |

(a)

|           |           |
|-----------|-----------|
| RManager1 | RManager2 |
| Task6     | Task5     |

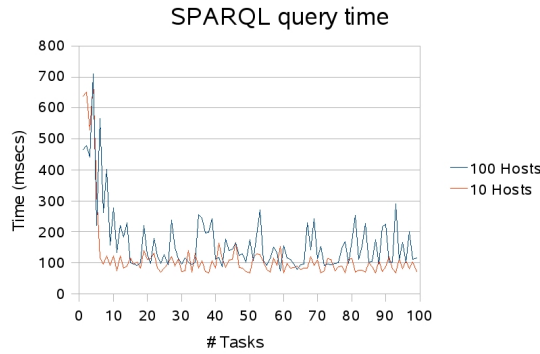
(b)

|                |           |
|----------------|-----------|
| RManager1      | RManager2 |
| Task6<br>Task5 | Task7     |

(c)

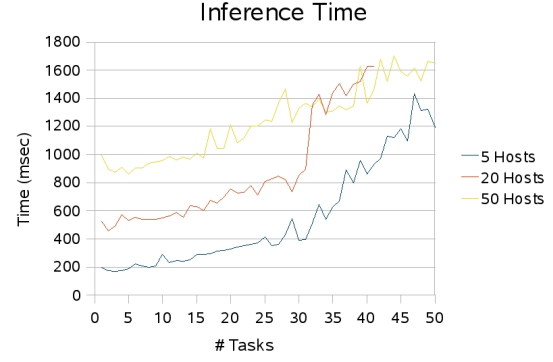
|                |           |
|----------------|-----------|
| RManager1      | RManager2 |
| Task6<br>Task8 | Task7     |

(d)

**Table 3. Task rescheduling****Figure 4. SPARQL query time**

ecution of the first queries. We also can see that the larger the number of hosts in the system is, the bigger the variations between the different queries are. On the other hand, the number of tasks in the system does not affect the query time. It was the expected behaviour because these sparql queries are only looking for hosts.

Regarding the inference time. We have made some measurements about the scheduling time Figure 5 shows the time of scheduling a task when there are 5, 20, and 50 hosts and different number of tasks. The Inference time includes the creation of the data model and inference with the rule engine. Two different zones can be distinguished in the figure. The first one is when the resources are not full and only the first rule is fired to perform the scheduling. The second zone is when the first rule is not fired and the 2nd and 3rd rules must to perform the scheduling. This second zone produce a faster increment of the scheduling time.

**Figure 5. Inference Time**

## 8 Conclusions and Future Work

This paper introduces a working framework for facilitating service provider management, which allows reducing costs and at the same time fulfilling the quality of service agreed with the customers. Our solution exploits the features of semantic technologies to perform the task scheduling taking into account task requirements as well as business parameters, the virtualization for building on-demand execution environments, and agents to coordinate the different components and react in case of failures.

In this paper, we describe a working implementation of the proposed architecture, which is part of the Semantically-Enhanced Resource Allocator prototype developed within the BREIN European project. We have focused in the part of the treatment of the semantically-enriched metadata in order to assign SP resources to client task in the way that the best clients are favored. We have proposed some extension to an existing resource ontology to fulfill our needs. We have also proposed a small set of inference rules that guide the inference process. This is the basis for our initial experiments described in this paper.

The behavior of the semantic scheduler is the expected, offering flexibility to re-schedule and cancel tasks to meet requests with more priority. The overheads of the match-making queries are significant at the initialization phase, but remain reasonable on the steady state. While the inference time grows linearly with the number of tasks, we consider this time still reasonable if we consider the granularity of the requested tasks.

As future work, we consider the extension of the ontology and of the set of rules that will take into account further business parameters. Additionally, we will study how inference overheads can be reduced by minimizing the amount of data considered by the rule engine applying previous filters.

## 9 Acknowledgments

This work is supported by the Ministry of Science and Technology of Spain and the European Union (FEDER funds) under contracts TIN2004-07739-C02-01 and TIN2007-60625 and the European Commission under FP6 IST contract 034556 (BREIN).

## References

- [1] EU BREIN project. <http://www.eu-brein.com>.
- [2] Common information model. <http://www.dmtf.org/standards/cim/>.
- [3] Ganglia Monitoring System. <http://ganglia.sourceforge.net/>.
- [4] The GLUE computing element schema. [http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/glue\\_ce.html](http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/glue_ce.html).
- [5] C. Goble, I. Kotsiopoulos, O. Corcho, P. Missier, P. Alper, and S. Bechhofer. Overview of S-OGSA: a Reference Architecture for the Semantic Grid. *Journal of Web Semantics*, 2006.
- [6] G. A. Grimnes, S. Chalmers, P. Edwards, and A. Preece. Granitenights - a multi-agent visit scheduler utilising semantic web technology. In *Seventh International Workshop on Cooperative Information Agents*, pages 137–151, 2003.
- [7] J. Ejarque and M. de Palol and F. Julià and I. Goiri and J. Guitart and R.M. Badia and J. Torres. Using Semantics for Enhancing Resource Allocation in Service Providers. Technical Report Research Report UPC-DAC-RR-2008-3, Computer Architecture Dept., UPC, 2007.
- [8] EU OntoGrid project. <http://www.ontogrid.net>.
- [9] W. Z. Paolo Missier and P. Wieder. Semantic Support for Meta-Scheduling in Grids. Technical report, CoreGRID TR-0030, 2006.
- [10] M. Papazoglou and D. Georgakopolous. Service-oriented computing. *Communications of the ACM*, 46(10):25–28, 2003.
- [11] EU Phosphorus project. <http://www.ist-phosphorus.eu>.
- [12] OGF Semantic Grid Research Group (SEM-RG). [http://www.ogf.org/gf/group\\_info/view.php?group=semrg](http://www.ogf.org/gf/group_info/view.php?group=semrg).
- [13] B. L. Smith, C. van Aart, M. Wooldridge, S. Paurobally, T. Moyaux, and V. Tamma. An Ontological Framework for Dynamic Coordination. In *Proc. of the Fourth International Semantic Web Conference*, 2005.
- [14] SPARQL query language for RDF. <http://www.w3.org/TR/rdf-sparql-query/>.
- [15] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based resource matching in the grid - the grid meets the semantic web. In *In Proceedings of the Second International Semantic Web Conference*, 2003.
- [16] Uniform interface to computing resources. <http://www.unicore.eu>.
- [17] N. N. V and K. S. Resource matchmaking in grid - semantically. In *The 9th International Conference on Advanced Communication Technology*, 2007.