Appeared in: CLOUD COMPUTING 2010 : The First International Conference on Cloud Computing, GRIDs, and Virtualization

Semantic Resource Allocation with Historical Data Based Predictions

Jorge Ejarque[*], Andras Micsik[‡], Raül Sirvent[*], Peter Pallinger[‡], Laszlo Kovacs[‡] and Rosa M. Badia[*][†]
[*]*Grid Computing and Clusters Group - Barcelona Supercomputing Center (BSC), Barcelona, Spain*
[†]*Artificial Intelligence Research Institute - Spanish National Research Council (IIIA-CSIC), Barcelona, Spain*
[‡]*Distributed Systems Department - Computer and Automation Research Institute*
Hungarian Academy of Sciences (MTA SZTAKI), Budapest, Hungary
{*jorge.ejarque, raul.sirvent, rosa.m.badia*}*@bsc.es*, {*micsik, pallinger, kovacs*}*@sztaki.hu*

***Abstract*—One of the most important issues for Service Providers in Cloud Computing is delivering a good quality of service. This is achieved by means of the adaptation to a changing environment where different failures can occur during the execution of different services and tasks. Some of these failures can be predicted taking into account the information obtained from previous executions. The results of these predictions will help the schedulers to improve the allocation of resources to the different tasks. In this paper, we present a framework which uses semantically enhanced historical data for predicting the behavior of tasks and resources in the system, and allocating the resources according to these predictions.**

***Keywords*-multi-agent, semantics, scheduling, resource allocation, historical data, predictions, grid computing, cloud computing, distributed systems.**

I. Introduction

The Service-Oriented Computing (SOC) paradigm [1]. relies on the composition of services to build distributed applications by using basic services offered by third parties. Those services are offered by service providers that create their description and their implementation. From the business point of view, the service provider agrees with its customers the Quality of Service (QoS) and level of service through a Service Level Agreement (SLA). The fulfillment or violation of the SLAs indicate the grade of satisfaction of customers with the Service Provider (SP), affecting directly or indirectly to the benefit of these provider. One of the most common SLA violation happens when unexpected events such as failures appear and the system is not able to adapt to this change.

Service adaptation is a current research topic, addressing the automatic reactions to unanticipated events. Adaptation mechanisms usually get active when something already went wrong. However, adaptation can be used also when we anticipate a problem to occur. Prediction mechanisms can help to warn about statistically probable unwanted situations, or we can just calculate the likelihood of certain service parameters. The simple aim is to learn from the past, in order to project events happened in the past to the future. There are several software toolkits supporting similar calculations in the area of data mining, etc.

In this paper, we introduce a generic framework for prediction and adaptation, and describe its application in a specific scenario (Cloud resource scheduling). The basic requirements for such an environment are:

- to provide generic capability of collecting log data about internal events,
- to unify the collected data so that global coherence can be revealed,
- to provide customizable methods for getting predictions based on the collected data,
- to feedback predictions into the realization of the scheduling and adaptation mechanisms.

This framework combines prediction techniques with semantic technologies, which introduces semantic knowledge to the data evaluated by predictors, and multi-agent systems, which introduce the pro-activity and distributed problem solving for increasing scalability, adaptability and self-management of the system. Predictions extracted from the semantic historical data are taken into account by a group of agents for allocating different customer's jobs in the most reliable resources.

The paper is organized as follows: Section II gives an overview of the architecture of our solution; Section III is focused on the implementation of job predictions and their usage in the resource allocation process; Section IV evaluates the framework; Section V compares our proposal with the related work; and Section VI concludes the paper.

II. Architecture

Figure 1 shows the architecture of our proposed framework. It depicts a resource allocator distributed across multiple agents based on the Multi-Agent Resource Allocation (MARA) approach [2] whose decision are based on predictions based on historical data. There are two differentiated parts in the architecture: the part which is related to the management of jobs and the part which is in charge of resource provisioning. The job management part allows the customers to make all the actions related to their jobs (submit, cancel, etc), while the resource provisioning part allows the system administrator to add and remove resources.

Both parts are built on top of a JADE agent platform [3]. The platform can be distributed across multiple locations deploying containers on each of them. Moreover, it

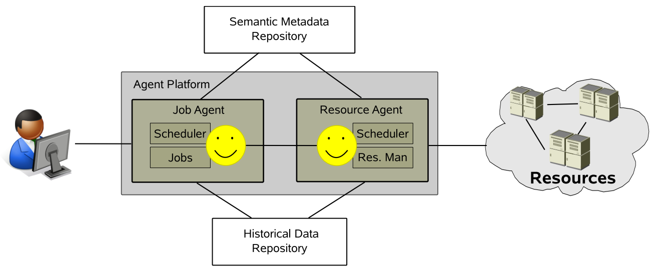


Figure 1. Semantic Resource Allocation with predictions

implements a messaging system, which allows the communication between agents located on different containers. The distributed configuration of the agent platform can improve the scalability of the system because the different parts can be processed in parallel on multiple hosts.

Customers jobs and resources are represented in the system by software agents. Job Agents are in charge of managing the customers jobs and Resource Agents are in charge of managing the providers resources. Moreover, the scheduling of the jobs in the different resources is made by an agreement reached from a negotiation between a Job Agent and different Resource Agents.

Apart from the job management and resource provisioning parts, the system architecture contains a Semantic Metadata Repository (SMR), which contains the semantic resource description registered in the system, and the Historical Data Repository (HDR), which contains semantically annotated logs from system events such as job executions, failures and other monitoring data, which is important to make predictions for current jobs. All the data stored in those semantic repositories is described according to a shared ontology providing a common framework for semantic data. The following paragraphs are focused on the most important part of the architecture.

A. *Scenario Ontology*

One of the most important issues about semantic technologies is the ontology, which models a common understanding of the concepts used on a scenario of study. In this case, we require an ontology for modeling the system entities such as customers, service providers, jobs and resources and other important data such as resource allocation data for job scheduling and historical data for making predictions.

The Grid Resource Ontology (GRO) [4] provides a model where the most important concepts and entities are described. This ontology provides a definition of the customer jobs and resources but it lacks concepts for describing resource allocation and historical data. Therefore, the GRO has been extended in the scope of the BREIN project [5] in order to cover the mentioned gaps.

The gap of resource allocation data in ontologies was studied in our previous work [6]. Resource requirements have been introduced in the GRO as a set of required abstract *GRO Resources* in the *GRO Task* definition, which models an abstract job. This definition has been also extended with time constraints such as expected duration, deadline, earliest possible start, etc. The scheduling result (assigned resources and time slot) has been introduced on the *GRO Process*, which incarnates the *GRO Task*. Required resources are linked to the task definition as resource sets. A resource set is typically a host, a container of more detailed resource descriptions (disk, CPU, etc.) with resource properties (e.g. size of disk). Furthermore, tasks are also linked to their representing agents and to related business information (such as customer data, service provider, agreed SLA) via the BREIN Business Ontology. These extensions provide the possibility for multi-scope description of job executions merging data about technical capabilities, schedules and business aspects into a single model. It is future work to fully exploit the new capabilities provided by this model. In this paper we provide the first steps in this direction.

GRO Process and *Task* descriptions are also useful for historical data. When a job has ended, the *GRO Process* description contains the result of this job (start time, end time, final status, ...) and the *GRO Task* (abstract job) contains the requested resources, the expected duration and the scheduled start and end times described. The GRO extension for describing historical data also contains classes for describing resources used by each task and resource downtimes. The comparison and evaluation of this data can be used for making predictions about how different types of jobs will run on different resources.

B. *System Agents*

Our system contains two types of system agents for managing jobs and resources. These agents have been implemented using a Belief-Desire-Intention (BDI) model [7]. For each type of agent, a set of data (Beliefs), goals (Desires) and plans (Intentions) are defined to model the behavior of the agent. Depending on the values of the data, events and the active goals on each moment, the BDI engine decides which plans has to execute the agent for reaching its goals. Our BDI agents have been developed with the Jadex framework [8] used on top of the JADE platform. The following paragraphs give more details about the job and resource agent functionalities and how the BDI model is applied for implementing them.

1) Job Agent (JA): The JA has the main goal of executing jobs in the resources of the system. This execution has different states, some of them indicate that the execution is running correctly and other ones indicate problems in the execution. Depending on the state of the job execution, the JA has to act in a different way. For this reason, the main goal of the JAs has been separated in several subgoals, which will be activated depending on the job status. The activation of subgoals triggers the execution of the plans to achieve them. For instance, when a new job execution is requested, the JA activates the goal for negotiating a resource allocation. In the *running* state, the JA activates the goal for monitoring the job execution evaluating if the required performance is fulfilled. Finally, if the job is *finished* the JA execute plans for deallocating the resource.

The JA has also to recover itself from status, which can alter the normal execution of the job (*stopped, suspended, non scheduled*). In those situations, the JA will execute plans to resubmit the job or to look for new resources.

2) Resource Agent (RA): The main goal of a RA is the management of resource capabilites for executing the jobs requested by the customer according to resource capabilities and the status and number of jobs assigned to the resources controlled by the agent (jobs *scheduled* and *running*). To provide this main feature, a set of subgoals and plans have been defined to negotiate resource allocations with JAs.

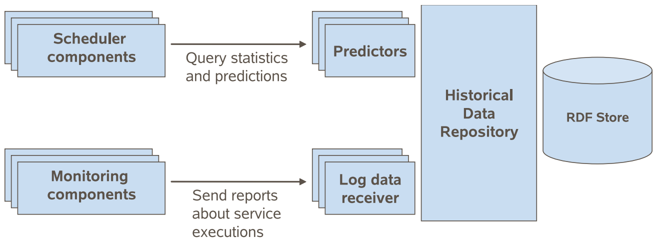
Apart from the negotiation subgoal, RAs contain two subgoals for monitoring the scheduled jobs and running jobs assigned to their resources. These subgoals are in charge of initiating the job execution depending on the planned start time or canceling a job if the deadline has been reached and new jobs are waiting for execution.

Additionally, the RA is also prepared to react to resource failures. The plan for recovering a failure sends a stopped notification message to JAs whose jobs were scheduled at the failed resources. Similarly, it also sends a suspended notification message to JAs whose jobs were already running. JAs treat these notifications according to the plans explained in the JA part.

C. The Historical Data Repository (HDR)

The HDR is a flexible, generic component implemented by SZTAKI to provide facilities for log data collection and on-demand predictions. The HDR is implemented in Java and can be embedded into Java code or can be run as a separate Web Service. In both cases, other software components can send their log data to the HDR, either by our customizable client APIs or via direct calls. Incoming information must be in the format of RDF [9], based on the core ontologies available to all components. The conversion to RDF can be implemented in client APIs if necessary. Thus, in our scenario, the service collects and stores information about past events (i.e., job executions and resource usage), and provides mining and searching for these mentioned past events.

The collected status information is stored as RDF inside the HDR. The repository can then be used for extracting statistics- and knowledge-based information or predictions. In its simple form, the repository can answer SPARQL queries to provide historic information. Clients can use the extended SPARQL provided by Jena ARQ interface [10], and ask for various aggregations of data, such as average, maximum, minimum. Additionally, Pellet [11] or the Jena built-in reasoner can be applied on the data for simple inferencing. For example, the use of subclasses can help to flexibly select a range of resource types for querying.


Scheduler components
Query statistics and predictions
Predictors
Historical Data Repository
RDF Store
Monitoring components
Send reports about service executions
Log data receiver
Figure 2. Historical Data Repository overview

For application-specific querying purposes a general plug-in mechanism has been developed. Predictors can be installed as plug-ins for HDR to answer specific questions. Within HDR, the RDF storage is coupled with the Weka data mining software [12]. In this way, a predictor can use both semantic querying and statistical methods. For example, a predictor can build a classification model on top of the results of a semantic query. The classifier can then be used to provide predictions based on the past.

In our specific scenario, the HDR is loaded with the description of the executed jobs, their resource usage and the resource downtimes. This data is used to train the classifier in order to provide estimations on the probabilities of delays in the schedule and problems with the job completion and estimations on the reliability of the used resources. As the statistical model is periodically updated, the provided values dynamically reflect the experiences of the recently finished executions.

D. Semantic Resource Allocation Process

The resource allocation for a particular job is decided between the JA and a set of RAs using the Contract Net Protocol (CNP) [13]. Figure 3 shows the message exchange between these agents for coordinating the job scheduling. When a JA has activated a goal for allocating resources to a job, it sends a query to the SMR to get the RAs whose resources match with the job requirements (1). Afterwards, the JA initiates a negotiation sending to the selected RAs a call for scheduling proposals (2). Each RA makes its own proposal and returns it to the JA (3). The JA evaluates all proposals, accepts the best for its interest and rejects the rest (4).

The RA proposals and the JA evaluations are done using an agent scheduler module. It is based on a rule engine evaluating a set of scheduling rules over semantic metadata bound to the GRO ontology (Section II-A). Despite of both agents using the same module, they behaves in a different way because they are loaded with different information and rules. The RA makes the scheduling proposals evaluating

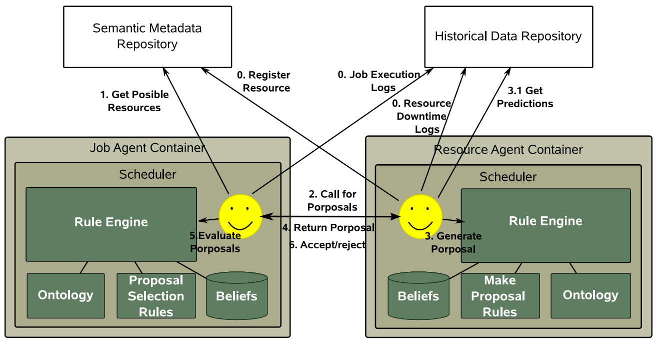


Figure 3. Distributed Semantic Scheduling

scheduling rules over the resource and assigned job descriptions, while the JA evaluates the scheduling proposal according to the customer rules and job description.

Once the RA has obtained a scheduling for a job in a resource, the RA consults the HDR to obtain predictions for this resource allocation (3.1). Resource allocation predictions are attached to the scheduling solution inferred by the scheduler to create a scheduling proposal, which will be evaluated by the JA. The following section gives more details about this process.

III. Job Prediction in Resource Allocation Process

In order to learn from past experiences, agents use the HDR as a service. In our HPC scenario, a separate HDR web service is applied for each Service Provider. This ensures that private internal data (such as log of execution details) remain within the boundaries of the provider, but also creates a merged knowledge base across various HPC clusters of the provider. A new plug-in was developed for the HDR, which is responsible for providing statistics and predictions for the scheduling agents. This predictor plug-in works on the basis of the common GRO explained in Section II-A containing the descriptions of hosts, software, host capabilities and QoS metrics.

When a new job is submitted to the system, it is described semantically indicating time constraints and the collection of resource requirements. During resource allocation process (described in Section II-D), the job is scheduled in the available resource and the results of this process are attached to the job description. During job execution, the JA monitors the execution introducing the relevant data in the job description (completion status, start time, end time and resource usage). Once the job has finished, the JA sends the job execution results to the HDR component using the HDR API client (Fig. 3, step 7). An example of this job execution report is shown in the following lines.

```
<rdf:Description rdf:about="gro:job-0">
  <gro:hasActionState rdf:resource="tech:done"/>
  <gro:hasResourceSet rdf:resource="gro:requirement_job-0"/>
  <rdf:type rdf:resource="gro:Execute_Task"/>
  <tech:hasDeadline>2010-06-14T19:41:07</tech:hasDeadline>
  <tech:expectedDuration>20</tech:expectedDuration>
  <gro:isExecutedAt rdf:resource="tech:host_1"/>
  <gro:incarnatedToProcess rdf:resource="gro:Incarnated_job-0"/>
</rdf:Description>

<rdf:Description rdf:about="gro:Incarnated_job-0">
  <rdf:type rdf:resource="gro:ProcessInstance"/>
  <tech:hasPlannedStart>2010-06-14T19:21:07</tech:hasPlannedStart>
  <tech:hasPlannedEnd>2010-06-14T19:41:07</tech:hasPlannedEnd>
  <gro:usesResource rdf:resource="tech:host_1"/>
  <gro:incarnatedFromTask rdf:resource="gro:job-0"/>
  <tech:hasActualStart>2010-06-14T19:21:07</tech:hasActualStart>
  <tech:hasActualEnd>2010-06-14T19:39:07</tech:hasActualEnd>
</rdf:Description>

<rdf:Description rdf:about="tech:usage_host_1_job-0">
  <tech:hasMeanMemoryUsage>25696.0</tech:hasMeanMemoryUsage>
  <tech:hasMeanCPUUsage>41720.0</tech:hasMeanCPUUsage>
  <tech:hasEnd>2010-06-14T19:39:07</tech:hasEnd>
  <tech:hasStart>2010-06-14T19:21:07</tech:hasStart>
  <tech:hasResource rdf:resource="tech:host_1"/>
  <tech:hasJob rdf:resource="gro:job-0"/>
  <rdf:type rdf:resource="tech:Usage"/>
</rdf:Description>
```

On the other hand, RAs report to the HDR about resource downtimes (Fig. 3, step 8). Job execution and resource downtime reports build up an extended log of actions inside the SP, which will be used as a knowledge base for generating the job predictions.

```
<rdf:Description rdf:about="tech:Downtime_host_1_1">
  <tech:hasEnd>2010-06-13T19:52:05</tech:hasEnd>
  <tech:isCausedBy rdf:resource="tech:powerOut"/>
  <tech:hasStart>2010-06-13T19:44:05</tech:hasStart>
  <tech:hasResource rdf:resource="tech:host_1"/>
  <rdf:type rdf:resource="tech:DownTime"/>
</rdf:Description>
```

Based on these data, a statistical classifier is trained, which can provide estimations on the probability of delays in the schedule, probability of problems with the job completion and on the reliability of the used resources. The relevant data for creating the statistical model is obtained from the HDR RDF store by means of SPARQL queries. For instance, we get the number of resource failures for the different types of resources and jobs from making the predictions of resource reliability and the job failure probability. The planned start and end times can be compared with the real start and end time for calculating delays on the scheduling and job duration. The statistical model is periodically updated with the historical data of recent execution. In this way, predictions dynamically reflect the experiences of past executions in the defined time window. The mechanism demonstrates the coupling of semantic data processing with data mining, as a promising novel combination.

Predictions are queried by the RA during the resource allocation process in order to make its scheduling proposal. The HDR provides the probability of delays in the job schedule (assigned host and time slot) inferred by the RA, the probability of problems during execution based on past executions with similar descriptions on similar hosts, and the reliability of the assigned host. The RA introduces these job predictions in the scheduling proposals and sends them to the

JA in order to be analyzed according to customer rules. In this case, the JA calculates reliability cost for the proposed scheduling based on a pondered mean of the predictions provided by the HDR. Once the job reliability cost for each proposal is analyzed, the JA will select the most reliable host, which fullfills the job constraints.

IV. Evaluation

In our experiments, we used the data from HPC resources of Barcelona Supercomputing Center (BSC). The test environment contained 8 hosts running 2 types of software. A HDR plug-in was implemented for the specific prediction tasks of the experiments. The plug-in at start-up extracts the necessary data using a SPARQL query:

```
SELECT ?task ?plannedStart ?plannedEnd ?start ?end
?resource ?status ?mem ?cpu WHERE{
  ?proc rdf:type gro:ProcessInstance .
  ?proc gro:usesResource ?resource .
  ?proc gro:incarnatedFromTask ?task .
  ?task gro:hasActionState ?status .
  ?proc tech:hasActualStart ?start .
  ?proc tech:hasActualEnd ?end .
  ?proc tech:hasPlannedStart ?plannedStart .
  ?proc tech:hasPlannedEnd ?plannedEnd .
  ?usage tech:hasJob ?task;
  ?usage tech:hasMeanMemoryUsage ?mem;
  ?usage tech:hasMeanCPUUsage ?cpu .
  FILTER (?start > "2010-01-01T00:00:00")
}
```

The data selected for this case includes the scheduled start and end for the job, the actual start and end times, the host where the job was run, the status of job completion and the mean memory and CPU usage of the job. Then, the necessary Weka data structure is built, and the classifiers are configured and run. Further data arriving during the predictor is in use can be added incrementally to the plug-in.

The expected duration to complete the job was calculated using various classifiers built into Weka. The best results were achieved by the M5P regression tree and the KStar instance-based classifier. The mean absolute error for the predicted value was 43 and 74 seconds respectively, where the actual value range was between 240 and 2400 seconds (i.e. the mean error is 3% of the mean predicted value).

Similarly, the status field can be used to predict the probability of failure. In this respect, Bayes-style, decision tree and decision rules algorithms were equally good at classifying the job completion status correctly for 85% of the job executions.

Furthermore, agents could assess the reliability of hosts by asking their availability metric from the plug-in. The availability is calculated based on the list of downtimes for the host. This function is supported directly by SPARQL queries, without using prediction mechanisms.

The time to load the predictor in the experiments can be broken up to the time needed for fetching the necessary data from the semantic log store (SPARQL query), and the time for building the prediction model with Weka. The first action required 1-2 seconds of time, while the second action could be accomplished in about 90 ms for 500 rows of data up to 141 ms for 5000 rows of data. As it was expected, the prediction time was independent of the number of data rows, yielding the result in approximately 15 ms.

We performed further testing of the prediction algorithms with log data available from two public archives: the Parallel Workloads Archive [14], and the Grid Workloads Archive [15]. The mean absolute error of the predictions in this case could be forced below 7% of the mean of the predicted value by careful preprocessing of the data. The comparison with our own data revealed that more details about job execution can yield better predictions in general. In contrast to regression used in cited related work (Section V), we preferred the instance-based learning algorithms, which relate similar job executions with each other for prediction purposes, and thus provided better predictions on job properties.

V. Related Work

Foster et al raised the benefit of integrating the results from agents and grids research areas in [16]. Agents could improve the autonomy, flexibility and scalabity of current grid systems. Regarding the area of resource allocation and job scheduling, the multi-agent system researchers have been already focusing on this problem. Several solutions have been proposed, such as the ones based on market-control where each agent tries to maximize its benefit function and the market controls them, the social wellfare where the multi-agent system tries to maximize a collective benefit and other proposals like game and decision theory. In the market-based solutions, we would like to highlight proposals like Challenger [17], Tycoon [18], other studies more focused on grids such as TRACE [19] or ARAM [20] and also projects such as CatNets [21] or SORMA [22]. The work on wellfare engineering and game theory for multiagents resource allocation has been compiled in [23] and [24] respectively.

All of these solutions implement specific allocation policies for solving a problem with their advantatges and drawbacks, but they are static and cannot be changed easily. In our paper we do not try to offer a new solution for the allocation algorithms, but we have introduced the multi-agent solution in the Semantic Resource Allocation process leaving users the availability of extending or changing the policies. In our system, customers and providers can describe the scheduling rules that are the most convenient for their interests. Those policies will be combined during the negotiation, trying to get a solution which satisfies all the policies.

In this case we have also introduced the capability of taking into account predictions based on semantic historical data about how similar jobs have been executed on the different resources. It allows the user taking into account in their policies the information provided by themselves as well as information about how the job execution is predicted by the system.

Predictions have been used also in other systems. The work in [25] describes an approach for predicting SLA values during the execution of WS-BPEL processes. At given points in the process, the collected QoS data are fed into a prediction engine, which provides predictions for numerical SLO values using neural networks. Predictions are presented on a graphical user interface and open facilities for manual interaction in the executing process. In [26], the authors present an architecture for event-based collection of historical data, and performing prediction on QoS data in a SOA environment. This approach does not use semantics and its focus on QoS is different than current paper.

VI. Conclusion

The paper emphasizes on the importance of using historical data by service and cloud providers. We present a generic approach and a re-usable solution for the collection and exploitation of historical log data produced by services. The heterogeneity of log data arriving from various resources calls for a semantic data representation, which can facilitate the unification of these data and a query mechanism supported by inference. Our approach demonstrates the coupling of semantic data processing with data mining as a promising novel combination.

Acknowledgment

This work is supported by the Ministry of Science and Technology of Spain and the European Union under contract TIN2007-60625 (FEDER funds), Generalitat de Catalunya under contract 2009-SGR-980 and the European Commission with FP6-IST project 34556 (BREIN) and FP7-ICT project 215483 (S-CUBE).

References

[1] M. Papazoglou and D. Georgakopolous, “Service-oriented computing,” *Communications of the ACM*, vol. 46, no. 10, p. 25, 2003.

[2] Y.Chevaleyre, et al, “Issues in Multiagent Resource Allocation,” *Informatica*, vol. 30, pp. 3–31, 2006.

[3] “Java Agent Development Framework,” http://jade.tilab.com[1].

[4] J. Brooke, D. Fellows, K. Garwood, and C. Goble, “Semantic matching of grid resource descriptions,” in *Grid Computing*. Springer, 2004, p. 240.

[5] BREIN Consortium, “Final Report on the BREIN Core Ontologies,” BREIN Project, Public Deliverable D3.2.5, 2008.

[6] J. Ejarque, et al, “Exploiting semantics and virtualization for SLA-driven resource allocation in SP,” *Concurrency and Computation: Practice and Experience*, vol. 22, no. 5, p. 541, 2010.

[7] A. Rao and M. Georgeff, “BDI agents: From theory to practice,” in *The 1st Int. Conf. on Multi-agent Systems*, 1995, p. 312.

[8] “Jadex system,” http://jadex.informatik.uni-hamburg.de[1].

[9] “Resource Description Framework,” http://www.w3.org/RDF[1].

[10] “ARQ - A SPARQL Processor for Jena,” http://jena.sourceforge.net/ARQ[1].

[11] “Pellet OWL reasoner,” http://clarkparsia.com/pellet/[1].

[12] I. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Pub, 2005.

[13] R. Smith, “The contract net protocol: High-level communication and control in a distributed problem solver,” *IEEE Trans. on Computers*, vol. 100, no. 29, pp. 1104–1113, 1980.

[14] “Parallel workload archive,” http://www.cs.huji.ac.il/labs/parallel/workload[1].

[15] “Grid workload archive,” http://gwa.ewi.tudelft.nl[1].

[16] I. Foster, N. Jennings, and C. Kesselman, “Brain meets brawn: Why grid and agents need each other,” in *The 3rd Int. Conf. on Autonomous Agents and Multiagent Systems*, 2004, p. 15.

[17] A. Chavez, A. Moukas, and P. Maes, “Challenger: A multiagent system for distributed resource allocation,” in *The 1st Int. Conf. on Autonomous Agents*, 1997, p. 331.

[18] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. Huberman, “Tycoon: An implementation of a distributed, market-based resource allocation system,” *Multiagent and Grid Systems*, vol. 1, no. 3, pp. 169–182, 2005.

[19] S. Fatima and M. Wooldridge, “Adaptive task resources allocation in multi-agent systems,” in *The 5th Int. Conf. on Autonomous Agents*, 2001, p. 544.

[20] S. Manvi, M. Birje, and B. Prasad, “An agent-based resource allocation model for computational grids,” *Multiagent and Grid Systems*, vol. 1, no. 1, p. 17, 2005.

[21] “CatNets Project,” http://www.catnets.uni-bayreuth.de[1].

[22] “Sorma Project,” http://www.sorma-project.eu[1].

[23] Y. Chevaleyre, U. Endriss, S. Estivie, and N. Maudet, “Welfare engineering in practice: On the variety of multiagent resource allocation problems,” *Engineering Societies in the Agents World V*, p. 335, 2005.

[24] S. Parsons and M. Wooldridge, “Game theory and decision theory in multi-agent systems,” *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 3, p. 243, 2002.

[25] P. Leitner, et al, “Runtime Prediction of SLA Violations for Composite Services,” in *3rd Workshop on Non-Functional Properties and SLA Management in SOC*, 2009.

[26] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, “Event-driven QoS prediction,” in *6th Int.Conf on SOC*, 2008, p. 147.

[1]Last access to links in the references is August 18th, 2010