Parallel, Distributed and Grid Computing for Engineering,
B.H.V. Topping and P. Iványi, (Editors),
Saxe-Coburg Publications, Stirlingshire, Scotland, 141–163.

Chapter 8

Resource Management in Grids: Overview and a Discussion of a possible Approach for an Agent-based Middleware

M. Senobari[1], M. Drozdowicz[2], M. Ganzha[2], M. Paprzycki[2,3], R. Olejnik[4], I. Lirkov[5], P. Telegin[6] and N.M. Charkari[1]

[1]*Tarbiat Modares University, Tehran, Iran*
[2]*System Research Institute Polish Academy of Sciences, Warsaw, Poland*
[3]*Warsaw Management Academy, Warsaw, Poland*
[4]*University of Sciences and Technologies of Lille, France*
[5]*Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria*
[6]*SuperComputing Center Russian Academy of Sciences, Moscow, Russia*

Abstract

Resource management and job scheduling are important research issues in computational grids. When software agents are used as resource managers and brokers in the Grid a number of additional issues and possible approaches materialize. The aim of this chapter is twofold. First, we discuss traditional job scheduling in grids, and when agents are utilized as grid middleware. Second, we use this as a context for discussion of how job scheduling can be done in the agent-based system under development.

Keywords: grid computing, agent-based resource management, resource management, job scheduling, metaschedulers.

8.1 Introduction

According to reference [1], a "computational grid is a hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to computational resources." In general, resources (also named services; see [2]) can include supercomputers, data repositories, clusters, sensors, workstations, programs, or individual PCs. However, in this chapter we are interested in the Grid as a collection of computational (rather than data / software) resources, which have to be effectively managed. However, the heterogeneous highly dynamic nature of the Grid makes development of Grid Resource Management Systems (GRMS) a challenge. There have been many projects focused on designing and implementing GRMSs. As has been argued in [3], the most important issues in the process are: (a) supporting adaptability, extensibility, and scalability, (b) allowing systems with different administrative policies to inter-operate while preserving site autonomy, (c) co-allocating resources, (d) supporting quality of service, and (e) meeting computational cost constraints.

In the case of grid computing, scheduling means allocation (matching) resources to incoming jobs (we use terms task and job interchangeably). While there exists exhaustive work on scheduling in "traditional" parallel and distributed systems, it is rarely applicable to the Grid. The main reason is that they often assume that tasks are completed utilizing homogeneous dedicated resources residing in a single administrative domain [3]. However, the vision of the Grid assumes that it consists of heterogeneous non-dedicated resources residing in multiple administrative domains.

One interesting approach to resource management in the Grid is based on software agents. While arguments presented in [4, 5] are not without critics, we are in the process of developing a GRMS, where agent teams are the basic managerial infrastructure [6]. Thus far we have concentrated our attention on interactions between users and the team leader (the *LMaster* agent) [7], and interactions between *Worker* agents and the actual Grid infrastructure [8]. Currently, we need to address the question: how will the *LMaster* select the *Worker* to execute a given job (for the time being we assume that a single *Worker* will execute a single job)? In this context, first, we present an overview of approaches to grid resource management and scheduling. Second, we discuss how understanding these approaches can help us to solve the load distribution problem within our agent-based infrastructure.

8.2 Grid resource managers and schedulers

An early survey of grid resource management systems was compiled in 2002 by Buyya *et al.* [3]. There, three basic approaches to job scheduling were distinguished: (i) centralized, (ii) hierarchical, and (iii) decentralized. The centralized scheduling is easiest to manage and deploy, but is not well suited for the Grid. Its main disadvantages are lack of scalability and fault-tolerance. In the hierarchical approach, schedulers at higher levels manage larger sets of resources (and utilize schedulers at lower levels). Hierarchical schedulers are utilized, for instance, in the 2K distributed operating sys-

tem [9], as well as in Darwin [10] and Legion [11] resource management systems. On the other hand, Ninf [12], MOL [13], and Bond [14] projects adopt decentralized scheduling. Note that Grid schedulers usually cannot control grid resources directly. They mostly work as *brokers* (or *agents*). However, grid schedulers can also be tightly coupled with applications (application-level scheduling; see [15]).

While grid heterogeneity includes applications, resources, as well as middleware components and their relations, it is possible to generalize processes involved in Grid scheduling. For instance, in [15] it is argued that grid scheduling consists of: (1) resource discovering and filtering, (2) resource selecting and scheduling according to certain objectives, and (3) job submission. In [15] we can find a birds-eye view of the structure of the grid resource management. Here, the grid scheduler (GS) receives jobs from grid users, performs matchmaking (utilizing an information service (IS)), and generates a schedule. In other words, the result of scheduling is a map between jobs and available resources. Note that in the grid multiple schedulers may exist in separate administrative domains, and may be organized in different structures. Delivery of a feasible schedule requires detailed information about (a) properties of the job and (b) of available resources, and (c) current state of the system. Here, the grid information service (GIS) provides system state information. After the schedule is prepared, submitting a task to selected resource(s) and monitoring execution is the responsibility of the Launching and Monitoring (LM) service. Another functionality of the LM is staging necessary executables and input data. Local Resource Manager (LRM) performs the final scheduling and submission inside a specific domain. The LRM has additional control over the available resources (*e.g.* it enforces administrative policies over resource usages, and reports the resource status information to the GIS). Let us now look at some details of popular schedulers.

8.2.1 AppLeS

The AppLeS project [16] was based at UC San Diego and was focused on development of scheduling agents for applications running in Grids. The AppLeS system collects resource information from the Network Weather Service (NWS) running at each computing node, and dispatches tasks to lighter loaded nodes; while scheduling actual execution of applications is local. AppLeS uses other RMSs, *e.g.* Globus, Legion, or NetSolve, to execute actual jobs (*i.e.* it can be viewed as a meta-middleware placed above the standard Grid middleware). Each application has embedded AppLeS agents that perform resource scheduling. AppLeS applications utilize templates that define specific computational models and thus allow reuse of application patterns (*e.g.* parametric and master-slave application templates have been provided). AppLeS facilitates predictive heuristic state estimation model, online rescheduling and fixed application-oriented scheduling policy [3, 17]. According to our best assessment the AppLeS project is no longer active. All of its publications and the WWW site were completed before 2001 and most of its participants have moved out of UC San Diego.

8.2.2 Nimrod/G

Nimrod/G is a Grid resource broker based on an economy-driven approach to manage resources and schedule jobs. It utilizes services provided by other Grid middleware (*e.g.* Globus, Legion, Condor), and the GRACE [18] trading mechanisms. More detailed explanation of economical methods used in Nimrod/G is presented in section 8.3.2. Note that, while at the Nimrod/G site there are references to work completed in 2007, the latest version of this middleware v3.0.1 was released in October, 2005. Thus, to the best of our knowledge, today this project is no longer active.

8.2.3 OpenPBS

OpenPBS [19,20] is a simple workload management solution intended for small clusters of dedicated homogeneous nodes. Here, computers are federated into a virtual pool of resources. Workload is scheduled to run within this virtual pool, based on simple scheduling algorithms. OpenPBS is one of workload managers accessible from the CSF metascheduler (see, Section 8.2.6). However, the last release of the OpenPBS as an independent project (v2.3.16) happened in 2001. At the same time, the PBS Professional is the commercial product developed and sold by the Altair corporation.

8.2.4 NetSolve

The NetSolve project [21] was focused on execution of scientific applications in heterogeneous environments, while utilizing different scheduling algorithms for different applications. Job completion time estimation was based on performance and load models, while a dynamic job queue was used for job ordering. The length of this queue was adaptively adjusted based on historical performance data (an example of a system-level scheduling [17]). In addition, mechanisms for scheduling multi-step, data-dependent jobs have been implemented [22]. Recently, the NetSolve project has been extended to Grids through the GridSolve infrastructure [23]. Both projects are active; for instance, a new release of GridSolve software appeared on 2008-12-18.

8.2.5 Condor

Condor [24] is a high-throughput computing environment that manages large collections of diversely owned machines. It utilizes a centralized scheduler based on the ClassAd matchmaker. To overcome the disadvantages of centralized scheduling, Condor allows the matchmaker (and/or the user) to forward requests to another matchmaker through the gateway flocking mechanism. The Condor project is still under development and has a large community of users.

8.2.6 Community Scheduler Framework

The Community Scheduler Framework (CSF) is an open source Web Services Resource Framework compliant [25] metascheduler built for the Globus Toolkit [26–28].

The CSF provides interface and tools for Globus users to create reservations, define scheduling policies and submit jobs to the Grid. CSF functionalities can be extended to utilize other schedulers and support different Grid deployment models. For instance, using CSF allows a single interface access to (i) Load Sharing Facility (LSF); [29]), (ii) OpenPBS, (iii) Condor, and (iv) Sun Grid Engine (SGE [30]). The CSF is the default metascheduler for the Globus Toolkit 4. This indicates that the CSF is not only active, but likely to be developed further (with the development of Globus).

8.2.7 ADAJ and SODAJ

The aim of the ADAJ (Adaptive Application in Java) and the SOADAJ (Service Oriented Adaptive Java Applications) projects is to develop an infrastructure to run applications in "desktop Grids" [31, 32]. ADAJ is a programming and execution environment, which contains mechanisms for dynamic re-distribution of components of applications (in response to load imbalances among Grid nodes). The SOAJA is a WSRF-based ([33]) service oriented extension of ADAJ. It is being developed on top of ADAJ, by adding the webservices layer. Furthermore, in SOAJA adjustments are being made to make it less dependent on proprietary solutions (*e.g.* the Enterprise Service Bus and the JavaParty); see [34] for more details. Similarly to ADAJ, SOADAJ is to facilitate workload measurement and component-level load balancing.

8.3 Approaches to task scheduling in grids

Job or task scheduling has a long history of research in parallel and distributed computing systems. Obviously, a large number of methods developed in that context have been adapted to grid scheduling. Since a thorough summary of grid scheduling methods can be found in [15, 35], we will focus our attention only on economic methods, which fit very well with agent systems (see literature related to utilization of software agents in e-commerce scenarios, for instance [36, 37]).

8.3.1 Economic models

Metaphorically speaking, computational grids and the power grid share the same general economic model. In the power grid, we use electricity and pay for usage of a "unit of electricity." In the computational grid we are to use computational resources and pay for their usage (possibly, one day a "unit of computing" will be established and globally accepted). Therefore, in the computational Grid we can distinguish producers and consumers (with their objectives) and commodities (resources) that are traded (*e.g.* programs, data, storage, CPU cycles, *etc.*). To support this aspect of the Grid (Grid economy), we need infrastructure (algorithms/policies) similar to that in the e-commerce. In other words, we need to be able to establish price of resource, service level agreement (SLA) and its enforcement, secure payment, *etc.* [17]. Similarly to the case of the (commodity) market, it is possible to utilize transactions based on both bartering and commodity prices. In the case of bartering, exchange of resources

takes place (*e.g.*, storage space for CPU time). When pricing is used, price of services should be based on supply, demand and possibly other economic factors.

For the grid economy to materialize, grid users need both: (1) to be able to specify resource requirements, and (2) to establish their preferences (*e.g.* that a given job should be high priority and price is not important). At the same time the Grid infrastructure (acting as a resource broker) has to be able to select resources that meet these requirements. Obviously, effective mechanisms for price / SLA negotiations are also needed. Mechanisms of this type have been studied in research in broadly understood e-commerce. For instance, when resource price is considered alone, mechanisms for establishing price equilibrium should be utilized.

According to [17], service providers and users can be functionalized as represented by grid service providers (GSPs) and grid resource brokers (GRBs). The GSPs deliver grid enabled resources (*e.g.* Globus or ADAJ) as well as grid trading services (GTS) to facilitate resource usage negotiations (based on user requests delivered by the GRBs). Interactions between GRBs and GSPs during contract negotiations are mediated by a Grid Market Directory (GMD).

Depending on the mechanism used for establishing transaction details (negotiating the service level agreement, the price, etc.) either GRBs or GSPs can initiate the negotiation. For instance, a GRB may invite proposals from GSPs' and select one that satisfies its requirements (*e.g.* job will be done before the deadline and within cost constraints). Alternatively, a GSP may invite bids from prospective users and offer its services to the highest bidder. Note that both GSPs and GRBs have constraints to be satisfied and utility functions to be maximized. Let us look into details of selected mechanisms that can be used to negotiate transaction details (for more information, as well as a list of projects that, before 2002, used economic models, see [17]).

8.3.1.1 Bargaining mechanism

In the bargaining model, GBRs bargain with GSPs, for instance for lower access price and/or higher usage duration. In the e-commerce literature this model is also known as *iterative bargaining* [38]. Here, GBRs and GSPs have objective functions and negotiate as long as their objectives are met or until it is established that finding an agreement is not possible. Negotiation can involve a single item (*e.g.* price), or multiple items (*e.g.* price and deadline). According to [17] this model is particularly useful when market supply-and-demand and service prices are not clearly established.

8.3.1.2 Tender/contract-net mechanism

The tender/contract-net model is one of the popular models for service negotiations (see also [39]). Here, the GRB announces its requirements (using a specific template) and invites bids from GSPs. Interested GSPs evaluate the announcement and respond by submitting their bids. The GRB awards the contract to the GSP that submitted the best offer. In the case when no (satisfactory) offer is obtained the GRB may adjust and resubmit its call for proposals. The negotiation template may include, among others,

addressee, requirements specifications (*e.g.* Linux, x86arch, and 1024MB memory), task/service description (*e.g.* a Matlab job), maximum price (optional), bid specification (what should offer contain), expiration time (deadline for receiving bids), *etc.* Note that the tender/contract-net allows for finalizing contracts without bargaining. This simplifies interactions and can improve the efficiency of the system (*e.g.* it is easy to imagine a very long sequence of iterations taking place in the case of bargaining).

8.3.1.3 Auction model

According to [17] the Auction model involves one-to-many negotiations, between a GPS and multiple GRBs, and reduces negotiation to a single value (*i.e.,* price). Note that this view is a clear oversimplification of what has been discussed in the e-commerce literature, see for instance [36, 40, 41]. Let us accept this view however as an approximation of what could be used in this case. Here, the basic process involves: (a) start of the Auction, (b) bid submission, (c) agreement formation, or establishing that agreement cannot be formed (see also [37]). Most popular forms of Auctions are: English Auction, First-price sealed-bid, Vickrey Auction, and Dutch Auction. However, the literature concerning single and multi-item auctions considers a much broader spectrum of contract negotiation mechanisms.

8.3.2 Job Scheduling in Nimrod/G

To illustrate application of an economic model in Grid resource management, let us look into the Nimrod/G approach. In [42], three economy-based algorithms used in the Nimrod/G are presented (note that neither one of them utilizes the elaborate economic mechanisms presented above):

- Time minimization—complete job(s) within time and budget constraints,
- Cost minimization—complete job(s) within time and budget constraints,
- None minimization—complete job(s) within time and budget constraints.

Let us assume that a task to be completed consists of one or more jobs. When the Time Minimization algorithm is used, the goal is to complete the task as quickly as possible (within the available budget). The key steps of this algorithm are as follows [42]:

1. For each available resource, use information about previously assigned jobs to estimate the completion time for a new job.
2. Sort resources according to the expected completion times.
3. Assign a given job to the resource for which the completion time is the "shortest," while the cost is less than the remaining budget per job.
4. Repeat until all jobs are assigned.

In the case of cost minimization, the goal is to complete the task as cheaply as possible, while still satisfying the deadline constraint ([42]):

- Sort available resources according to the advertised cost (cheapest first).
- Utilize estimates of job execution times to assign as many jobs as possible to cheapest resources; without exceeding the deadline.

Finally, for the none minimization algorithm, the goal is to complete the task within the deadline and cost constraints, but no minimization of either is attempted [42]:

- Divide resources in such a way that in each case cost per job is less than budget available per job.
- In the case of the cheaper resources, assign jobs (inversely) proportionally to the estimated job completion time (*e.g.* "cheap resource" with estimated completion time $= 4$ received twice as many jobs as resource with estimated completion time $= 8$).
- In the case of more expensive resources, repeat steps (until all jobs are assigned), but use recalculated budget per job (budget based on money left after cheaper resources have been contracted).

Note that implementation of either one of these three strategies involves taking care of special situations. For instance, during systems startup completion time estimates are unknown, and thus a supplemental strategy has to be used. Similarly, when there are "too many" jobs in the task and they cannot be *all* assigned to available resources (an example of an infeasible schedule), the system has to be able to manage such situation.

8.4 Agent based scheduling systems

Since in our project we utilize software agents as Grid resource managers, let us now focus our attention on approaches to grid resource scheduling appearing in the context of agent-grid integration.

8.4.1 ARMS

ARMS was an agent-based grid RMS. It used agents for resource advertisement and discovery [43]. It utilized performance prediction mechanisms provided by the Performance Analysis and Characterise Environment (PACE) toolkit [44]. Furthermore, scheduling was focused on QoS requirements; *e.g.* users had to specify an explicit job execution deadline.

In ARMS homogeneous cooperating agents are organized in a hierarchy, and their goal is to manage and schedule applications over available Grid resources. Each agent acts as a representative for a single Grid resource, while PACE is used to create a hardware characterization template. Next, hardware model and services information is spread across the agent hierarchy. This information is utilized to build the Agent Capability Table (ACT). Each ACT item is a tuple containing the agent ID, information about services as well as performance. ACTs are updated periodically and both pull and push methods are used to maintain them.

When an application is submitted to the system, it includes an associated performance model and requirements related to its execution (*e.g.* start time, deadline, *etc.*). Service discovery involves communicating with neighboring agents in the hierarchy (upward and downward). It is claimed that this approach is therefore more scalable when the Grid becomes large.

ARMS agents consist of three layers: (i) communication layer, (ii) coordination layer, and (iii) local management layer. The communication layer acts as an interface to the external world. It receives service advertisements and discovery messages and forwards them to appropriate modules in the coordination layer, which perform matchmaking and scheduling. For the service discovery message, the agent tries to find an available Grid resource. This involves utilization of (a) the application model, (b) job requirements, and (c) the PACE engine. Specifically, the expected execution time of a given application on a selected resources is estimated and compared with the requirements. If time constraints are satisfied for one of them, the process is completed. Otherwise the agent forwards the request to other agents (higher or lower in the hierarchy) to find resource that will satisfy user defined constraints.

In [45], an ant-based self-organizing mechanism is utilized to perform load balancing for batch jobs with no explicit execution deadlines. It is shown that application of such mechanism improves global load balancing in the system (given a large enough number of ants). Separately, in [46], scheduling dependent jobs and executing workflows in ARMS were discussed.

It should be noted, that while this approach seems very interesting, work on the PACE framework and the ARMS system has not been pursued further since approximately 2003. What is left are only papers reporting results (no agent code pertinent to any part of the system can be found).

8.4.2 JADE extensions

In their work, Poggi *et al.* [47] extended the JADE agent framework to be used in grid applications . They argued that realizing an agent-grid integration is possible through: (i) extending grid middleware to support agent features, or (ii) extending agent-based middleware to support functionalities of the Grid. They follow the second approach by attempting to add new features to JADE to realize a Grid environment. Considered extensions are mechanisms for: code distribution, reconfiguration, goal delegation, load balancing optimization, and QoS definition. To realize these goals new types of agents are proposed. They are to support: (i) rule-based creation and composition of tasks, and (ii) mobility of the code at the task level (*i.e.,* JADE behaviors or rules are exchanged by agents). First, a *Drools agent* is developed, which uses the Drools rule engine, to receive and execute rules (coming from other agents). The *BeanShell agent* receives and executes behaviors coming from other agents. It integrates the BeanShell engine, which allows usage of Java as a scripting language. Here, each rule can contain scripts in its condition, consequence or extractor fields. When a rule is scheduled for execution (its preconditions are satisfied), Drools invokes the BeanShell interpreter to execute the code contained in the consequence of the rule. To address

security issues in the Grid, the JadeS security framework is used.

Work on extending JADE agent framework was originally reported in 2004. Since then the following three papers, in some way related to the subject, have been published [48–50]. It is thus difficult to see this project as being actively pursued. This is even more so, as there is no Grid related add-on listed among JADE add-on software [51].

8.4.3 Bond

Bond is a Java-based object-oriented middleware for network computing. Bond was developed to create an infrastructure for a virtual laboratory. It was to support scheduling of complex tasks and data annotation for data intensive applications. One of the goals of the Bond system was to facilitate collaborative activities through support for knowledge and workflow management. These functionalities are based on a distributed object system [52] (*e.g.* to store and process resource information). Resources exchange information using messaging and utilizing the KQML language. Dissemination of resource information is achieved through periodic data pushes. In Bond, a mechanism called distributed awareness is used to learn about the existence of other agents. Specifically, each node maintains information about nodes it has communicated with, and periodically exchanges it with other agents. In this way, information about existing agents is propagated in the system. Finally, job scheduling is decentralized and utilizes predictive pricing models for state estimation [52].

Again, this situation is similar to the two other systems described thus far. All publications related to the Bond system have appeared in the 1999–2003 time frame. Therefore, we have to conclude that the project is no longer active.

8.4.4 Agent-based scheduling framework

Agent-based Scheduling Framework (ASF) is an agent-centered scheduling approach applied to Grid scheduling [53]. ASF is composed of a metascheduler and autonomous agents attached to computing resources. The main idea of the ASF is to reduce the responsibilities of a conventional metascheduler. Specifically, the main issues that the ASF attempts to address are ([53]):

- The workload of the metascheduler grows when the number of computing resources grows.
- If the metacheduler is overloaded, accuracy of scheduling degrades as it is going to be based on incomplete information (not all information can be processed in time to make accurate predictions).
- What is needed is a new approach to managing large numbers of heterogeneous computers; especially, when many domains with various operation policies are combined in a grid as a VO (conventional frameworks are not ready to deal with such complex issues).

To achieve its goals the ASF relies on agents that autonomously search for jobs, instead of jobs being assigned to them by the metascheduler. Thus, the ASF is "dual" to conventional metaschedulers, which continuously collect state information of resources under their control, decide about the scheduling policy and *push* jobs to selected resources. In the ASF, instead, each agent discovers jobs that can be processed by its resource and retrieves them from the metascheduler (a *pull* based approach).

A prototype of the ASF has been implemented based on the Globus Toolkit 4. Experimental results of utilization of the ASF metascheduler have been discussed in [53]. It was shown, that for a relatively small environment (a heterogeneous cluster with three nodes(!)) utilization of the ASF resulted in an 11% reduction of the total elapsed time of job processing.

8.4.5 MAGDA

Mobile agent based grid architecture (MAGDA; [54]) is a Java-based mobile agent toolkit designed to overcome some of the limitations of existing Grid middlewares. According to its creators [54], these include:

- Lack of ability to migrate an application from one system to another.
- Low level of abstraction of the heterogeneity of the environment.
- Lack of robust fault tolerance.
- Existing information and monitoring frameworks do not scale to the grid level (or are focused only on specific issues).
- Lack of support of task migration, monitoring and execution (with adequate checkpointing).

MAGDA supports (1) resource discovery, (2) performance monitoring and load balancing, and (3) task execution within the Grid. It has a layered architecture following the Layered Grid Model [55] and thus it should be possible to implement or integrate MAGDA components with other systems that are based on the same model.

In the current release of MAGDA, service discovery is performed with the help of web services and Web Services technologies (such as UDDI) are used to implement this component. Application-level load balancing is provided by means of a *Coordinator Agent*.This agent manages lists of registered *Workers* and of available free hosts. When an agent is initialized, coordinator updates the list of registered agents. It stores information about their state of computation, and their relative speed. In this way, the *Coordinator Agent* can recognize imbalance in the system and ask the most loaded *Worker* to split its workload, part of which will then be assigned to the less loaded *Worker*.

After migration from Aglets [56] to JADE [51], MAGDA is continually being developed with the aim of maximization of integration with Web Services, workflow management, service orchestration and choreography [57].

8.5 Job scheduling in the *Agents in Grid* Project

Let us now look at how issues discussed above can be utilized in the *Agents in Grid* project. In our approach it is assumed that agents work in teams [6]. Each team is managed by the *LMaster* agent. This agent also represents the team to the outside world. In other words, agents representing *Users*, named *LAgents* interact with *LMasters* either to contract job execution [58], or to join the team and to become a *Worker* [7]. Each *LMaster* is supported by an *LMirror* agent, which stores copies of crucial team data (*e.g.* list of workers, list of contracts, status of job execution, *etc.*). In the context of this chapter, we are interested in processes that take place when a job is contracted and forwarded to a selected *Worker* to be actually executed.

8.5.1 Forming the Team

To start we need to consider the way that workers join the team. As described in [7], the process consists of the following steps: (a) *User* specifies conditions of joining to its *LAgent*, (b) the *LAgent* interacts with the *Client Information Center*, represented by the *CIC* agent, to obtain a list of teams that seek workers satisfying certain conditions (*e.g.* CPU power, available memory. *etc.*), (c) upon receiving such list, the *LAgent* eliminates these that are not trustworthy [59], (d) next it utilizes the FIPA Contract Net Protocol [60] to negotiate which team to join (note that, in the context of this chapter, we omit details of contract negotiations; *e.g.* how the *LMaster* establishes the optimal price; we simply assume that one of the bargaining mechanisms described in Section 8.3.1 is used), (e) negotiations can result in a success (joining a team), or in a failure (no acceptable team was found). Here, we are particularly interested in: (i) description of capabilities of resource(s) represented by the *LAgent*, and (ii) details of the contract. To illustrate them, let us consider the ACL message in Figure 8.5.1. It could have been sent by the *LAgent* as a call for proposals (within the Contract Net Protocol).

Here, agent (*demi*), representing machine (*barszcz*) with CPU running at 2.6 GHz, 2 Gbytes of RAM and 8 Gbytes of disc space available for grid applications, is sending a *take-me* message to agent *bruce* (*LMaster* of some team residing on node *tequila*). Agent *demi* is proposing the following contract conditions: availability each day between 22:00 and 7:55 (next day in the morning), and contract duration 14 days.

It is obvious that this is the information that matches what is utilized by the above described schedulers. First, provided are capabilities of the machine (Grid node) represented by agent *demi*. They are needed to be able to estimate job completion time. Second, let us assume that *bruce* "takes" *demi* to be a *Worker* in the team. In this case, contract details allow *bruce* to approximate if a task contracted by its team can be completed while the grid node that agent *demi* represents will still be available. It would be very bad for the reputation of the "bruce-team" if a job was to be paused until the "demi-node" comes back online, and as a result contract conditions would not be fulfilled. Observe that this form of utilization of available information follows the Nimrod/G-based approach.

```
(cfp
  :sender (agent-identifier :name demi@barszcz:1099/JADE)
  :receiver (agent-identifier :name bruce@tequila:1099/JADE)
  :content
    ((action
      (agent-identifier :name bruce@tequila:1099/JADE)
      (take-me
        :configuration (hardware
          :cpu 2.6
          :memory 2048
          :quota 8000)
        :conditions (condition
          :availability (frequency
            :unit (day)
            :day-time (period
              :from 00000000T22000000
              :to 00000000T07550000))
          :contract-duration +00000014T000000000))
  :language fipa-sl0
  :ontology joining-ontology
  :protocol fipa-contract-net
)
```


Figure 8.1: Sample call for proposals, containing description of available resources and conditions of joining

8.5.2 Contracting job execution

The second aspect that needs to be considered is: what happens when *User* would like a job to be executed in our system? As described in [58] the process consists of the following steps: (a) *User* specifies to its *LAgent* acceptable conditions of job execution, (b) the *LAgent* contacts the *CIC* agent to obtain the list of teams that have the required resources, (c) this list is then adjusted on the basis of trust considerations [59], (d) as in the previous case, FIPA Contract Net protocol is utilized as a negotiation mechanism to find the best team to execute the task, (e) process can result in a success (finding a team to do the job), or in a failure (no acceptable team is found). Here, of particular interest is available information concerning the job. This information is a part of the Contract Net CFP. To illustrate our initial approach let us present a snippet of our ontology of constraints:

```
:NegotiationSet a owl:Class .
:negotiationParam a owl:ObjectProperty ;
    rdfs:domain :NegotiationSet ;
    rdfs:range NegotiationParam .
:NegotiationParam a owl:Class .
:paramWeight
    a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :NegotiationParam ;
    rdfs:range xsd:float .

:Cost a owl:Class ;
    rdfs:subClassOf :NegotiationParam .
:costConstraint
    a       owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:domain :Cost ;
    rdfs:range :FloatConstraint .
:costValue
```

```
    a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :Cost ;
    rdfs:range xsd:float .
:JobStartTime a owl:Class ;
    rdfs:subClassOf :NegotiationParam .
:jobStartTimeValue
    a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :JobStartTime ;
    rdfs:range xsd:dateTime .
:jobStartTimeConstraint
    a owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:domain :JobStartTime ;
    rdfs:range :TimeConstraint.
:JobEndTime a owl:Class ;
    rdfs:subClassOf :NegotiationParam.
:jobEndTimeValue
    a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :JobEndTime ;
    rdfs:range xsd:dateTime .
:jobEndTimeConstraint
    a owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:domain :JobEndTime ;
    rdfs:range :TimeConstraint .
```


Here, we can see that three constraints of job execution were conceptualized: (i) *cost* (as a constraint, this value is private to the *LAgent*, but is also used in the contract), (ii) *job start time*, and (iii) *job end time*. It is obvious, that *job start time* and *job end time* may provide useful information for job schedulers, but this needs to be considered further. Note that it is possible to use the *job start time* constraint also in the case when there is no estimate as to how long this job is going to take. However, in this case we cannot have the *job end time* specified as well, as there is no feasible way to estimate if this constraint can be satisfied. Furthermore, in this case (*job start time* only), job has to be scheduled on a node with non-stop contract for an extended time (*e.g.* constant contract). Utilization of the *job end time* requires existence of at least some estimate of the job execution time. This information may originate from: (a) job description (could be provided by the *User* as a part of the CFP), or (b) could be based on past execution times. However, the latter case requires that information about the "type of the job" has to be a part of the CFP. Only then the *LMaster* would be able to estimate job execution time utilizing historical data. These considerations show clearly that the question of description of jobs, resources, constraints has to be revisited. Being aware of this, in another chapter of this book [61], we have presented an overview of resource descriptions utilized in various grid systems, as well as attempts at creating an ontology of the Grid. As a result we have found that the core grid ontology is the one that is most likely going to be utilized in our system. However, we have established also that it will have to be extended to include additional aspects of resource management, brokering and scheduling, necessary for our system (e.g. to deal with various constraints discussed above and support trust management).

Let us now assume that an appropriate grid ontology has been selected and extended to facilitate robust descriptions of: (1) resources, (2) job characteristics, (3) job execution constraints, and (4) trust related concepts. It should be obvious that such ontology will support advanced contract negotiation scenarios. In this context let us stress that

following [1,2], we believe that one of the key aspects of future grid computing will be the economic model (involving *Users* who pay for job execution, or are paid for usage of their resources). This being the case, we consider economic mechanisms described in Sections 8.3.1 and 8.3.2 as necessary for high-level resource management in the Grid. In this context, we expect that development of a robust Grid ontology will allow us to utilize semantic reasoning and thus opens up the Grid to a large number of autonomic contract negotiations mechanisms considered, among others, in e-commerce research (see, also [36, 62–64]). However, we will omit the very process of contract negotiations as outside of scope of this chapter.

Now, let us see what happens when a CFP reaches the *LMaster*. The first thing that it has to do is to check job execution constraints. Let us assume that on the basis of available data, the *LMaster* (i) can obtain an estimate of job execution time; or (ii) establishes that there is a contradiction in constraints—open ended job (no completion time estimate available) combined with the *job end time* constraint; or (iii) finds out that it has to deal with an open ended job. In the case of constraint contradiction, the *LMaster* can send an ACL *REFUSE* message to the CFP originator. In the remaining two cases the *LMaster* can combine the job and constraint information with its knowledge of: (a) all "job execution contracts" that have been as signed thus far, (b) current "work contracts" of all team *Workers*, and (c) current status of team *Workers* (which *Workers* are available, what is their workload *etc.*) to decide whether the proposed job can be executed within specified constraints or not. For instance, the *LMaster* should not contract an open ended job if it does not have a trustworthy *Worker* with non-stop contract. Similarly, it cannot contract a job with deadline in 24 hours, if all of its *Workers* are 100% utilized for the next 36 hours. In such cases, the *LMaster* responds with an ACL *REFUSE* message. If job execution is feasible (*i.e.*, constraints can be satisfied), then the *LMaster* sends a "positive" response to the originator of the CFP. This response may contain a detailed final proposal (*e.g.* in the case of contract net negotiations) or may be a part of more elaborate contract negotiations.

Recall that we assume that *Worker* agents work within their contract and are paid on its basis and thus do not posses access to the information on what price the *LMaster* charges for their work (however, they may explore offers from multiple teams to find the market price of resources they offer). In this way the sole responsibility for contract negotiation is on the *LMaster*. Obviously, in a different scenario it could be assumed that the *LMaster* would negotiate "subcontracts" with its *Worker*s (*e.g.* use a local Contract Net protocol) and use results of these (sub)negotiations to prepare a response to the *User*. However, we consider this approach unnecessarily complicated and stay with the model in which the *LMaster* is the only contract negotiator.

8.5.3 Selecting worker to execute a task

Thus far we have established that during the process of accepting team members (*Workers*), the *LMaster* obtains a complete description of resources represented by each one of them. Furthermore, for each *Worker* it knows details of its contract. Additionally, during evaluation of the call for proposals, the *LMaster* is able to establish that its team is capable of fulfilling the request within specified constraints. Let us now assume that the *LMaster* was able to successfully complete contract negotiations. Now, it has to obtain all information/data necessary to complete the job (including, for instance, needed files or information about their locations; see, [8], for more details). Recall that the *LMaster* is the *only* representative of the team known to the outside world. Here we follow basic tenets of agent system design [65] and reduce the number of possible agent interactions (*i.e. LAgents* representing *Users* do not know individual *Workers*). Assuming that the *LMaster* has all the information ready for the job execution to be initiated, the following two questions arise: (1) which agent should actually execute the job and (2) when should it receive it?

In general, we have identified three possible responses to the first question. The first one is based on the ASF approach (see, Section 8.4.4). Here, the *LMaster* acts as a metascheduler, while *Workers* contact it to obtain the next task to be executed. While this approach has some potential advantages (*e.g.* reducing the workload of the *LMaster* and giving *Workers* more autonomy), it does not seem to work well in the case of Grid economy, as it is not able to handle complex SLA's. For instance, let us assume that a special job-contract has been successfully negotiated. This task requires specific resources and immediate commencement of execution. In return, price is substantially higher than a typical one. In this case the *LMaster* cannot wait until the right resource becomes available and requests that job (recall, that in the ASF, agents pull tasks from the metascheduler "at their will"). Instead, the *LMaster* should be able to "act" and rearrange work of the team in such a way that the high-priority high-paying job would start executing immediately while using the right resources. This illustrates that the ASF approach is not easy to combine with the economic model.

The second possible approach is conceptually based on [66, 67]. Here the *LMaster* and *Workers* negotiate job execution. Note that, as specified above, we assume that the *LMaster* is the sole contract negotiator, while *Workers* act in a non-competitive way. As a matter of fact, they should support each other, as the success of the team means also their (financial) success. Obviously, this assumption may need to be changed (introducing agent spoilers and/or competitive/selfish behaviors of *Workers*), but this would require reexamination of all trust-focused considerations (see [59]) and thus, for the time being, will not be pursued further. In the negotiation-based approach, when the *LMaster* wins a contract, it could utilize the FIPA Contract Net to inform *Workers* and to find out which of them could execute it. In response to the CFP, *Workers* would inform about their conditions of job execution. Answers received from each *Worker*

could be evaluated on the basis of their resources, contracts and trust to establish which should execute the given job. Here, trust is used to attempt to avoid (or at least to minimize the impact of) failures of *Workers* (see, [59] for more details). While in [66, 67] improvement of the order of 10% resulting from utilizing this method has been reported, we can see also some problems with this approach. Basically, this is the problem discussed in the previous solution. Let us assume that a high-paying job has been negotiated and has to start immediately. In the current scenario it is possible to envision that as a result of internal negotiations this job may be started immediately on an appropriate machine. The selected *Worker* stops executing its current job to work on the special task. However, this situation may have a domino effect. Now, the job that has been executed on that machine may need to be moved to another one (it also has a relatively high priority and a close deadline), and so on. Now, it is extremely difficult to envision how such job movement can be achieved in the situation when each *Worker* is an autonomous entity that may or may not agree to the change. And, even if they do agree, the process of job-alignment may involve a large number of additional negotiations, or will actually be realized using the third approach (discussed in the next paragraph). Overall, in this approach, autonomy given to *Workers* may turn against the capability of the team to efficiently complete some jobs, and thus compete in the marketplace.

Finally, the *LMaster* itself can decide about the task assignment without any communication with its *Workers*. In other words, the *LMaster* can act as a metascheduler. This being the case, following examples of metaschedulers described above, the *LMaster* can use historical performance data for each *Worker*, information about their current load, *etc.*, to establish which resource should execute which job. We can assume here that the *LMaster* not only can decide which *Worker* will execute which job, but also can issue an *order* that a given *Worker* should send its job to another, while the recipient has to accept it. In other words, we functionalize the *LMaster* as an omnipotent metascheduler. It is easy to observe that this approach minimizes communication between the *LMaster* and *Workers* and overall programmatic complexity of interactions within the team; only orders are send to *Workers*, who report their completion. Furthermore, this means that *Workers* do not have to have special reasoning capacities (required in the second approach, in particular). Finally, this approach provides an easy way to deal with jobs that come at various priorities and deadlines (it is the *LMaster* that takes care of them). Unfortunately, we can find also an important disadvantage: requirements placed on the *LMaster* increase considerably as it has to be able to deal with all possible scenarios without any "help" from *Workers*. This, in turn, substantially increases hardware requirements of the node it runs on. This requirement propagates also to the *LMirror*, which has to be as good as the *LMaster* as it may take its place at any moment. Furthermore, the *LMaster* may become the bottleneck of the team (it will not scale with increasing team-size), which was one of the important disadvantages of the centralized scheduling noted above. However, our proposed system is to be adaptive. Thus, the *LMaster* that cannot handle the load will loose both clients and *Workers*. As a result its team will either decrease in size (to the size it can manage

successfully), or disappear completely. We can thus assume that, as the time passes, each *LMaster* will be able to establish the size of the team it can manage. The question that remains open is, will the size of the team be large enough for this approach to be feasible in a long run? However, an answer to this question can be established only experimentally. Summarizing, we believe that the advantages of this approach outweigh its potential disadvantages. We are thus likely to pursue this approach as the first attempt at introducing job scheduling into agent teams in our system.

As far as the second question is concerned (when should the job be transferred to the selected *Worker*) taking into account the dynamic nature of the Grid, there is no easy answer to it. First, recall that we assume that each node can disappear without warning. This means that staging a job at node X as soon as it is contracted may result in a wasted effort if this node crashes. However, since the *LMaster* can crash as well, it means that data needs not only to be kept until it is released to the selected *Worker*, but also has to be mirrored by the *LMirror*. We will thus leave this question open, pending further analysis.

8.5.3.1 Monitoring in the system

One of the important issues in most schedulers is monitoring the workload. In the case of our system not only the workload of *Workers* has to be monitored. More importantly, their very existence needs to be established. Recall that we assume that Grid nodes can disappear without any warning. However, monitoring existence of nodes has been addressed in [68]. There, we have reported how we have implemented a mechanism (based on principles derived from network management) in which the *LMaster* is "pinging" its *Workers* and expects that they respond in time to a certain number of pings. Lack of response is an indication of a problem with the *Worker*.

Let us now assume that the *LMaster* monitors existence of team members. What it needs is information about their workloads. Since Java does not have a direct method to access the load information of the underlying system, an external library had to be used to fulfill this requirement. *Jsysmon* is a Java library (which works both in Linux and Windows) that permits Java applications to access system monitoring information, *e.g.* the CPU or the Memory usage [69]. This library is used by the *Worker* agent. Specifically, in the *Worker* agent, we have added a new behavior (*UsageReporterBehaviour*), which has been implemented as a subclass of the standard JADE *TickerBehaviour*. At predefined intervals, this behavior collects the load information from the local system (using the *Jsysmon* library commands), and sends it as an ACL message to the *LMaster*. At the same time, the *LMaster* has been extended by adding a workload data collecting behavior (*MonitorStatusBehaviour*), which is a subclass of standard JADE *CyclicBehaviour*, which receives the load messages from team members and stores them for future use.

8.6 Concluding remarks

The aim of this chapter was three-fold. First, to present a brief overview of grid resource management techniques found in standard Grid middlewares. Second, to consider attempts at utilizing software agents as a Grid middleware, and thus as resource brokers and job (meta)schedulers. Finally, to discuss how the knowledge gathered in the first two parts of the chapter influences our thinking about job scheduling within our system.

We have realized, again, that the time has come to infuse our system with a robust grid ontology (see also, [61]). This ontology will not only allow us to utilize a broad range of possible SLA negotiation mechanisms. It will also provide the scheduler with the necessary information about the resources, the job and its execution constraints. Finally, analysis of available scheduling techniques performed within the context of our system (based heavily on the grid economy based vision of the nature of grid computing of the future) pointed out that at this stage we should proceed with making the *LMaster* an omnipotent manager and metascheduler, which has total command over *Workers* in its team. We will report on our progress in subsequent publications.

Acknowledgments

Work of the Polish team was in part supported from the "Funds for Science" of the Polish Ministry for Science and Higher Education for years 2008-2011, as a research project (contract number N516 382434). Collaboration of the Polish and Bulgarian teams is partially supported by the *Parallel and Distributed Computing Practices* grant. Collaboration of Polish and French teams is partially supported by the PICS grant *New Methods for Balancing Loads and Scheduling Jobs in the Grid and Dedicated Systems*. Collaboration of the Polish and Russian teams is partially supported by the *Efficient use of Computational Grids* grant.

References

[1] I. Foster, C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure", Morgan Kaufmann Publishers Inc., 1999.

[2] K. Czajkowski, I. Foster, C. Kesselman, "Agreement-based resource management", 93(3), 631–643, 2005.

[3] K.K. Rajkumar Buyya, M. Maheswaran, "A taxonomy and survey of grid resource management systems for distributed computing", Software Practical Experience, 32(2), 135–164, 2002.

[4] I. Foster, N. Jennings, C. Kesselman, "Brain Meets Brawn: Why Grid and Agents Need Each Other", International Joint Conference on Autonomous Agents and Multiagent Systems, 1, 8–15, 2004.

[5] H. Tianeld, R. Unland, "Towards self-organization in multi-agent systems and

Grid computing", Multiagent and Grid Systems, 1(2), 89–95, 2005.

[6] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, M. Paprzycki, "Utilizing agent teams in grid resource management—preliminary considerations", Proc. of the IEEE J. V. Atanasoff Conference, 46–51, IEEE CS Press, Los Alamitos, CA, 2006.

[7] W. Kuranowski, M. Paprzycki, M. Ganzha, M. Gawinecki, I. Lirkov, S. Margenov, "Agents as resource brokers in grids—forming agent teams", Proc. of the LSSC Meeting, Volume 4818, LNCS, Springer, Berlin, 2007.

[8] M. Senobari, M. Drozdowicz, M. Paprzycki, W. Kuranowski, M. Ganzha, R. Olejnik, I. Lirkov, "Combining an JADE-agent-based Grid infrastructure with the Globus middlewareInitial Solution", in M. Mohammadian (Editor), "Proc. of the CIMCA-IAWITC 2008 Conference", 890–895, IEEE CS Press, Los Alamitos, CA, 2008.

[9] F. Kon, R. Campbell, M. Mickunas, K. Nahrstedt, F. Ballesteros, "2K: A Distributed Operating System for Dynamic Heterogeneous Environments", HPDC'00: Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, 201–210, IEEE Computer Society, Washington, DC, USA, 2000.

[10] P. Chandra, A. Fisher, C. Kosak, T. Ng, P. Steenkiste, E. Takahashi, H. Zhang, "Darwin: Customizable Resource Management for Value-Added Network Services", IEEE International Conference on Network Protocols, 177–188, IEEE Computer Society, Los Alamitos, CA, USA, Oct 1998.

[11] S. Chapin, D. Katramatos, J. Karpovich, A. Grimshaw, "The Legion Resource Management System", Job Scheduling Strategies for Parallel Processing, 162–178, 1999.

[12] H. Nakada, M. Sato, S. Sekiguchi, "Design and Implementation of Ninf: Towards a Global Computing Infrastructure, Future Generation Computing Systems (Metacomputing Special Issue)", 1999.

[13] J. Gehring, A. Streit, "Robust Resource Management for Metacomputers", HPDC, 105–112, 2000, `citeseer.ist.psu.edu/gehring00robust.html`.

[14] L. Boloni, D. Marinescu, "An object-oriented framework for building collaborative network agents", Intelligent systems and interfaces, 31–64, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[15] F. Dong, S.G. Akl, "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems", Technical report, Queen's University School of Computing, January 2006.

[16] "About AppLeS", `http://www.cs.ucsd.edu/groups/hpcl/apples/hetpubs.html#AppLeS`.

[17] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, "Economic models for resource management and scheduling in Grid computing", Concurrency and Computation: Practice and Experience, 14(13–15), 1507–1542, 2002.

[18] "GRACE—GRid seArch & Categorization Engine", `http://www.grace-ist.org/`.

[19] "OpenPBS homepage", http://www.openpbs.org/.

[20] "PBS Pro homepage", http://www.pbsgridworks.com/.

[21] J. Dongarra, "NetSolve: A network server for solving computational science problems", The International Journal of Supercomputer Applications and High Performance Computing, 11(3), 212–223, 1997.

[22] M. Beck, H. Casanova, J. Dongarra, T. Moore, J. Plank, F. Berman, R. Wolski, "Logistical quality of service in NetSolve", Computer Communications, 22(11), 1034–1044, 1999.

[23] "NetSolve(GridSolve) Overview", http://icl.cs.utk.edu/netsolve/overview/index.html.

[24] "Condor Manual", http://www.cs.wisc.edu/condor/manual/.

[25] "The WS-Resource Framework".

[26] "CSF", http://www.globus.org/grid_software/computation/csf.php.

[27] C. Smith, "Open source metascheduling for Virtual Organizations with the Community Scheduler Framework (CSF)", Technical Whitepaper, 2003.

[28] L. Ferreira, M. Batista, S. Fibra, C.Y. Lee, C.A.Q. Silva, J. Almeida, F. Lucchese, N. Keung, Grid Computing Products and Services, IBM Redbooks, 2005.

[29] "Load Sharing Facilitiy", http://help.unc.edu/4484.

[30] "Sun Grid Engine Project", http://www.sun.com/software/gridware/.

[31] R. Olejnik, E. Laskowski, B. Toursel, M. Tudruj, I. Alshabani, "DG-ADAJ: a Java Computing Platform for Desktop Grid", in K.W. Marian Bubak, M. Turala, (Editors), "Cracow Grid Workshop '05 Proceedings", Academic Computer Centre CYFRONET AGH, Cracow, Poland, April 2006.

[32] E. Laskowski, M. Tudruj, R. Olejnik, B. Toursel, "Bytecode Scheduling of Java Programs with Branches for Desktop Grid", Future Generation Computer System (FGCS) - The International Journal of Grid Computing: Theory, methods and Applications, 23(8), 977–982, November 2007.

[33] R. Prodan, T. Fahringer, "From Web Services to OGSA: Experiences in Implementing an OGSA-based Grid Application", Grid, 2, 2003.

[34] M. Ganzha, M. Paprzycki, I. Lirkov, "Trust Management in an Agent-based Grid Resource Brokering System—Preliminary Considerations", in M. Todorov, (Editor), "Applications of Mathematics in Engineering and Economics'33", Volume 946, AIP Conf. Proc., 35–46, American Institute of Physics, College Park, MD, 2007.

[35] Y. Li, Z. Lan, "A Survey of Load Balancing in Grid Computing", Computational and Information Science, 280–285, 2005.

[36] C. Badica, M. Ganzha, M. Paprzycki, "Implementing Rule-Based Automated Price Negotiation in an Agent System", Journal of Universal Computer Science, 13(2), 244–266, 2007.

[37] C. Bartolini, C. Preist, N. Jennings, "Architecting for Reuse: A Software Framework for Automated Negotiation", Proceedings of AOSE, Volume 2585, 88–100, Springer, Berlin, 2002.

[38] M. Paprzycki, M. Ganzha, "Adapting Price Negotiations to an E-commerce System Scenario", in K. Saeed, *et al.*, (Editors), "Proceedings of the CISIM Conference", 380–386, IEEE CS Press, Los Alamitos, CA, 2007.

[39] "FIPA Contract Net Interaction Protocol Speci?cation", `http://www.fipa.org/specs/fipa00029/SC00029H.html`.

[40] B.Q. Li, J.C. Zeng, M. Wang, G.M. Xia, "A negotiation model through multi-item auction in multi-agent system", 2003 International Conference on Machine Learning and Cybernetics, Volume 3, 1866–1870, 2003.

[41] H. Benameur, B. Chaib-draa, P. Kropf, "Multi-item auctions for automatic negotiation", Journal of Information and Software Technology, 44, 291–301, 2002.

[42] R. Buyya, J. Giddy, D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter weep Applications", Proceedings of the Second Workshop on Active Middleware Services (AMS 2000), Kluwer Academic Press, Pittsburgh, USA, August 2000.

[43] J. Cao, "ARMS: An agent-based resource management system for grid computing", Scientific Programming, 10(2), 135–148, 2002.

[44] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, D. Wilcox, "Pace—A Toolset for the Performance Prediction of Parallel and Distributed Systems", Int. J. High Perform. Comput. Appl., 14(3), 228–251, 2000.

[45] J. Cao, "Self-organizing agents for grid load balancing", Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), 2004.

[46] J. Cao, S.A. Jarvis, S. Saini, G.R. Nudd, "GridFlow: Workflow Management for Grid Computing", 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'03), 2003.

[47] A. Poggi, M. Tomaiuolo, P. Turci, "Extending JADE for Agent Grid Applications", Proc. of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2004.

[48] A. Negri, A. Poggi, M. Tomaiuolo, "Intelligent Task Composition and Allocation through Agents", 255–260, 2005.

[49] A. Negri, A. Poggi, M. Tomaiuolo, P. Turci, "Dynamic Grid tasks composition and distribution through agents", Concurrency and Computation: Practice and Experience, 18(8), 875–885, 2006.

[50] A. Poggi, M. Tomaiuolo, P. Turci, "An Agent-Based Service Oriented Architecture", Proc. of the WOA'07, 157–165, 2007.

[51] "Homepage of the JADE project", `http://jade.tilab.com/`.

[52] L. Boloni, K. Jun, K. Palacz, R. Sion, D. Marinescu, "The Bond Agent System and Applications", Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, Volume 1882, 99–112, Springer-Verlag, London, UK, 2000.

[53] K. Sakamoto, H. Sato, "A Resource-Oriented Grid Meta-Scheduler Based on Agents", Proc. of the 25th IASTED International Multi-Conference, Parallel and Distributed Computing and Networks, Innsbruck, Austria, 2007.

[54] R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque, "MAGDA: A Mobile Agent based Grid Architecture", Journal of Grid Computing, 4(4), 395–412,

2006.

[55] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal of High Performance Computing Applications, 15(3), 200–222, 2001.

[56] "Wikipedia: Aglets", `http://en.wikipedia.org/wiki/Aglets`.

[57] R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque, "A skeleton based programming paradigm for mobile multi-agents on distributed systems and its realization within the MAGDA Mobile Agents platform", Mobile Information Systems, 4(2), 131–146, 2008.

[58] M. Dominiak, M. Ganzha, M. Paprzycki, "Selecting grid-agent-team to execute user-job—initial solution", Proc. of the Conference on Complex, Intelligent and Software Intensive Systems, 249–256, IEEE CS Press, Los Alamitos, CA, 2007.

[59] M. Drozdowicz, M. Ganzha, W. Kuranowski, M. Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, M. Senobari, I. Lirkov, "Software Agents in ADAJ: Load Balancing in a Distributed Environment", Applications of Mathematics in Engineering and Economics'34, 527–540, 2008.

[60] "Welcome to the FIPA", `http://www.fipa.org/`.

[61] M. Drozdowicz, M. Ganzha, M. Paprzycki, R. Olejnik, I. Lirkov, P. Telegin, M. Senobari, "Ontologies, Agents and the Grid: An Overview", in B.H.V. Topping, P. Ivanyi, (Editors), "Parallel, Distributed and Grid Computing for Engineering", Chapter 7, Saxe-Coburg Publications, Stirling, UK, 2009.

[62] B. Schnizler, "Resource Allocation in the Grid, A Market Engineering Approach", PhD thesis, 2007.

[63] Z. Tan, "Market-Based Grid Resource Allocation Using A Stable Continuous Double Auction", PhD thesis, 2007.

[64] J. Li, R. Yahyapour, "Negotiation Strategies for Grid Scheduling", Advances in Grid and Pervasive Computing, 42–52, 2006.

[65] M. Wooldridge, "An Introduction to Multiagent Systems", John Wiley & Sons, Chichester, England, 2002.

[66] H.J. Burckert, K. Fischer, G. Vierke, "Holonic Transport Scheduling with Teletruck", 'Applied Artificial Intelligence, 14(7), 697–725, August 2000.

[67] S. Bussmann, K. Schild, "An Agent-Based Approach to the Control of Flexible Production Systems", Proc. of the 8th IEEE Int. Conf. on Emergent Technologies and Factory Automation (ETFA 2001), Volume 2, 481–488, IEEE CS Press, Los Alamitos, CA, 2001.

[68] W. Kuranowski, M. Ganzha, M. Paprzycki, I. Lirkov, "Supervising Agent Team an Agent-based Grid Resource Brokering System—Initial Solution", in F. Xhafa, L. Barolli, (Editors), "Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems", 321–326, IEEE CS Press, Los Alamitos, CA.

[69] "Jsysmon—JAVA library for system monitoring", `http://jsysmon.sourceforge.net/`.