

# A Multi-agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing<sup>1</sup>

D. Ouelhadj<sup>1</sup>, J. Garibaldi<sup>2</sup>, J. MacLaren<sup>3</sup>, R. Sakellariou<sup>4</sup>,  
and K. Krishnakumar<sup>5</sup>

<sup>1,2</sup> School of Computer Science and IT, University of Nottingham,  
Jubilee Campus, Nottingham, NG8 1BB, UK

{dxs, jmg}@cs.nott.ac.uk

<sup>3,4,5</sup> University of Manchester, Oxford Road, Manchester, M13 9PL, UK  
{jon.maclaren, rizados, krish}@man.ac.uk

**Abstract.** In this paper we propose a new infrastructure for efficient job scheduling on the Grid using multi-agent systems and a Service Level Agreement (SLA) negotiation protocol based on the Contract Net Protocol. The agent-based Grid scheduling system involves user agents, local scheduler agents, and super scheduler agents. User agents submit jobs to Grid compute resources. Local scheduler agents schedule jobs on compute resources. Super scheduler agents act as mediators between the local scheduler and the user agents to schedule the jobs at the global level of the Grid. The SLA negotiation protocol is a hierarchical bidding mechanism involving meta-SLA negotiation between the user agents and the super scheduler agents; and sub-SLA negotiation between the super scheduler agents and the local scheduler agents. In this protocol the agents exchange SLA-announcements, SLA-bids, and SLA-awards to negotiate the schedule of jobs on Grid compute resources. In the presence of uncertainties a re-negotiation mechanism is proposed to re-negotiate the SLAs in failure.

## 1 Introduction

Grid computing has emerged as a new paradigm for next generation distributed computing. A Computational Grid is an infrastructure which enables the interconnection of substantial distributed compute resources in order to solve large scale problems in science, engineering and commerce that cannot be solved on a single system [6]. One of the most challenging issues from a Grid infrastructure perspective is the efficient schedule of jobs on distributed resources. Scheduling jobs in a Grid computing environment is a complex problem as resources are geographically distributed having different usage policies and may exhibit highly non-uniform performance characteristics, heterogeneous in nature, and have varying loads and availability.

---

<sup>1</sup> This work is funded by the EPSRC Fundamental Computer Science for e-Science initiative (Grants GR/S67654/01 and GR/S67661/01), whose support we are pleased to acknowledge.

The Grid scheduling problem is defined as: “given appropriate input, the high performance scheduler determines an application schedule, which is an assignment of tasks, data, and communication to resources, ordered in time, based on the rules of the scheduling policy, and evaluated as performance efficient under the criteria established by the objective function. The goal of the high-performance scheduler is to optimise the performance [1].

Within the Grid community at present, there is a keen focus on the management and scheduling of workflows, i.e. complex jobs, consisting multiple computational tasks, connected either in a Directed Acyclic Graph (DAG), or in a more general graph, incorporating conditionals and branches [15]. In order for a workflow enactment engine to successfully orchestrate these workflows, it must be possible to schedule multiple computational tasks onto distributed resources, while still respecting any dependence in the workflow. Current methods employed to schedule work on compute resources within the Grid are unsuitable for this purpose. Traditionally, these scheduling systems are queue based and use simple heuristics such as First-Come-First-Served (FCFS), or more complicated and efficient methods such as Backfilling, Gang Scheduling, Time Slicing, etc [9]. Such batch scheduling systems provide only one level of service, namely ‘run this when it gets to the head of the queue’, which approximates to ‘whenever’. New patterns of usage arising from Grid computing have resulted in the introduction of advance reservation to these schedulers, where jobs can be made to run at a precise time. However, this is also an extreme level of service, and is excessive for many workflows, where often it would be sufficient to know the latest finish time, or perhaps the soonest start time and latest end time. Advance reservation (in its current form) is also unsuitable for any scenario involving the simultaneous scheduling of multiple computation tasks, either as a sequence, or tasks that must be co-scheduled. Also, advance reservation has several disadvantages for the resource owner. When an advance reservation is made, the scheduler must place jobs around this fixed job. Typically, this is done using backfilling, which increases utilisation by searching the work queues for small jobs, which can plug the gaps. In practice, this rarely works perfectly, and so the scheduler must either leave the reserved processing elements empty for a time, or suspend or checkpoint active jobs near to the time of the reservation. Either way, there are gaps in the schedule, i.e., CPU time which is not processing users’ work. As utilisation often represents income for the service owner, there is a tendency to offset the cost of the unused time by charging for advance reservation jobs at a considerably higher tariff. As these new patterns of usage increase, utilisation will fall further. While it is possible to set tariffs high enough to compensate, this brute-force solution is inefficient in terms of resources, and undesirable for both users, who pay higher prices, and for resource owners, who must charge uncompetitive prices.

Recently, Multi-Agent Systems [5, 14] and the Contract Net Protocol [18] have proven a major success in solving dynamic scheduling problems and have given answers to the problem of how to efficiently integrate and cooperate communities of distributed and interactive systems in a wide range of applications [16]. A multi-agent system is a network of agents that work together to solve problems that are beyond their individual capabilities [14]. Multi-agent systems have been known to provide capabilities of autonomy, cooperation, heterogeneity, robustness and reactivity, scalability, and flexibility [5, 16]. A number of initiatives to apply agents in

computational Grids have appeared [2, 7, 11]. Most these agent-based Grid scheduling systems are centralised and static as scheduling is performed by a Grid high-performance scheduler (broker), and resource agents do not use any flexible negotiation to schedule the jobs. In a Grid environment resources are geographically distributed and owned by different individuals. It is not practical that a single point in the virtual system retains entire Grid's information that can be used for job scheduling and cope with the dynamic nature of the Grid environment. Therefore, the use distributed scheduling and negotiation models would be more efficient, flexible, and robust.

In this paper we propose a fundamental new infrastructure for efficient job scheduling on the Grid based on multi-agent systems, and a Service Level Agreement (SLA) negotiation protocol based on the Contract Net Protocol [13]. Section 2 describes SLAs. Section 3 presents the proposed SLA based Grid scheduling multi-agent system infrastructure. Section 4 elaborates the SLA negotiation protocol. Section 5 describes the SLA re-negotiation process in the presence of failures. Conclusions are presented in Section 6.

## 2 Service Level Agreement

Service Level Agreements (SLAs) [3, 10] are emerging as the standard concept by which work on the Grid can be arranged and co-ordinated. An SLA is a bilateral agreement, typically between a service provider and a service consumer. These form a natural choice for representing the agreed constraints for individual jobs. While there are technologies for composing SLAs in XML-based representations, e.g. WSLA [12], these embed domain-specific terms; no terms for resource reservation have yet been proposed within the Grid community. In any case, it is certain that SLAs can be designed to include acceptable start and end time bounds and a simple description of resource requirements. SLAs expressing conventional requirements of “at time HH:MM” or “whenever” could still be used where necessary, although these—especially the latter—may not be accepted by the scheduler. The GGF GRAAP Working Group [8] is interested in SLA terms for resource reservation, but has not yet put forward any designs for what these SLAs should look like. It is intended that the project will feed back a design of these SLA terms to the community, contributing to the standardisation process, and influencing the development of emerging technologies, such as the negotiation protocol WS-Agreement, which is being defined by the GGF GRAAP Working Group.

## 3 The Grid Scheduling Multi-agent System Infrastructure

The multi-agent infrastructure proposed is organised as a population of cognitive, autonomous, and heterogeneous agents, for integrating a range of scheduling objectives related to Grid compute resources. Figure 1 shows the multi-agent infrastructure for SLA based Grid scheduling. The infrastructure involves three types of agents: User agents, Local Scheduler (LS) Agents, and Super Scheduler (SS) Agents. Three databases are also used in this architecture to store information on SLAs, LS agents, and resources.

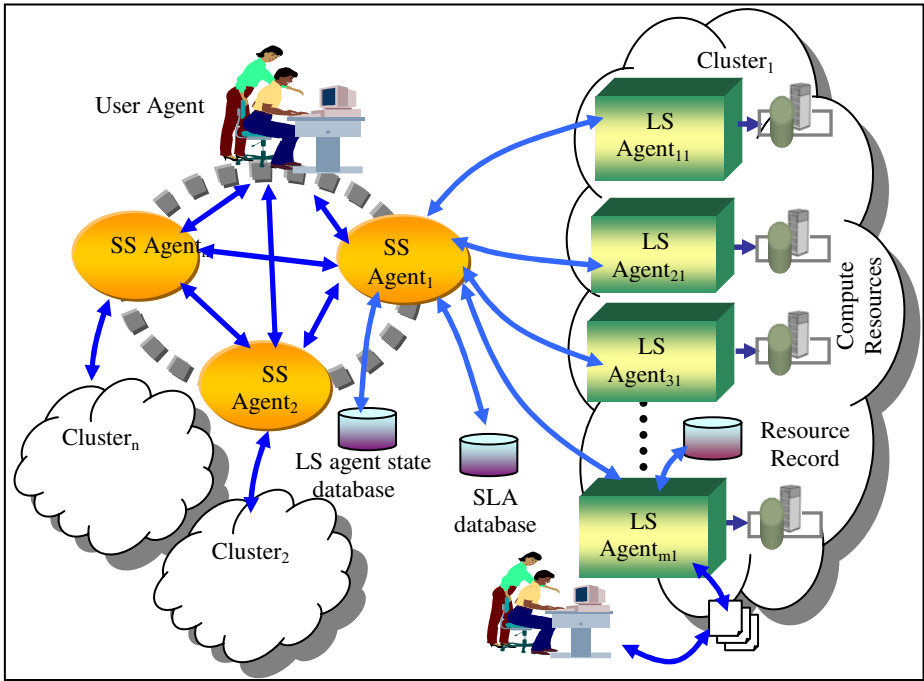


Fig. 1. SLA based grid scheduling system infrastructure

**User Agent:** The user agent requests the execution of jobs on the Grid and negotiates SLAs with the SS agents. The user agent can also submit jobs locally to SL agents.

**Local Scheduler Agent:** Each computer resource within each institution is assigned to an LS agent. The set of compute resources and their corresponding LS agents constitute a cluster. The LS agents are responsible for scheduling jobs, usually assigned by the SS agents, on compute resources within the cluster. However, jobs cannot only be arriving from the SS agents, but they can also be injected from other means such as PC which is locally connected to the machine that hosts the LS agent.

**Super Scheduler Agent:** SS agents are distributed in the Grid having each at every institution, and act as mediators between the user agent and the LS agents. Each cluster of LS agents of the same institution is coordinated by an SS agent. The user agent usually submits jobs to the SS agents. SS agents negotiate SLAs with the user agent and the LS agents to schedule the jobs globally on compute resources.

**Databases:** The databases used in the architecture are the following:

- *Resource record:* holds information about the compute resources on which the jobs are executed by an LS agent.
- *SLA database:* stores the description of the SLAs.
- *LS agent state database:* holds the status of LS agents and the loading/usage information about their local resources which is used by the SS agents.

## 4 SLA Negotiation Protocol

We propose an SLA negotiation protocol based on the Contract Net Protocol to allow the SS agents to negotiate with the user agent and the LS agents the schedule of jobs on the distributed computer resources. The Contract Net Protocol is a high level protocol for achieving efficient cooperation introduced by Smith [18] based on a market-like protocol. The basic metaphor used in the Contract Net Protocol is, as the name of the protocol suggests, contracting. It is the most common and best-studied mechanism for distributed task allocation in agent-based systems [5, 16]. In this protocol, a decentralised market structure is assumed and agents can take on two roles: a manager and a contractor. The manager agent advertises the task by a *task announcement* to other agents in the net. In response, contracting agents evaluate the task with respect to their abilities and engagements and submit *bids*. A bid indicates the capabilities of the bidder that are relevant to the execution of the announced task. The manager agent evaluates the submitted bids, and selects the most appropriate bidder to execute the task, which leads to *awarding* a contract to the contractor with the most appropriate bid. The advantages of the contract net protocol include the following: dynamic task allocation via self-bidding which leads to better agreements; it provides natural load balancing (as busy agents need not bid); agents can be introduced and removed dynamically; and it is a reliable mechanism for distributed control and failure recovery.

The proposed SLA negotiation protocol is a hierarchical bidding mechanism involving two negotiation levels:

**1. Meta-SLA negotiation level:** In this level the SS agents and the user agents negotiate meta-SLAs. A meta-SLA contains high-level description of jobs supplied by the user and may be refined or additional information may be added in the final agreed SLA at the end of negotiation. A meta-SLA may have the following baseline information:

- Resource information: the required capability of the resource for the application. It is usually very high level description of a resource metric containing resource details such as machine information, the range of processors, etc.
- Estimated due date: the time by which the job should be finished.
- Estimated cost: it is the maximum cost limit pre-set by the user for any given job execution request.
- Required execution host information (optional).

Uncertainty in user requirements such as time and cost constraint is represented by fuzzy numbers [17, 19].

**2. Sub-SLA negotiation level:** In this level the SS agents negotiate sub-SLAs with the LS agents. The SS agents decompose the meta-SLA into its low level resource attributes, sub-SLAs which contain low level raw resource description such as processes, memory processors, etc. A sub-SLA may have the following baseline information:

- Low level raw resource descriptions such as number of nodes, architecture, memory, disk size, CPU, network, OS, application (if any needed).
- Time constraint.
- Cost constraint.
- Storage Requirement.

### 4.1 Steps of the Hierarchical SLA Negotiation Protocol

The hierarchical SLA negotiation protocol involves several negotiation steps at each level of the protocol. The steps are described below and illustrated in figure 2.

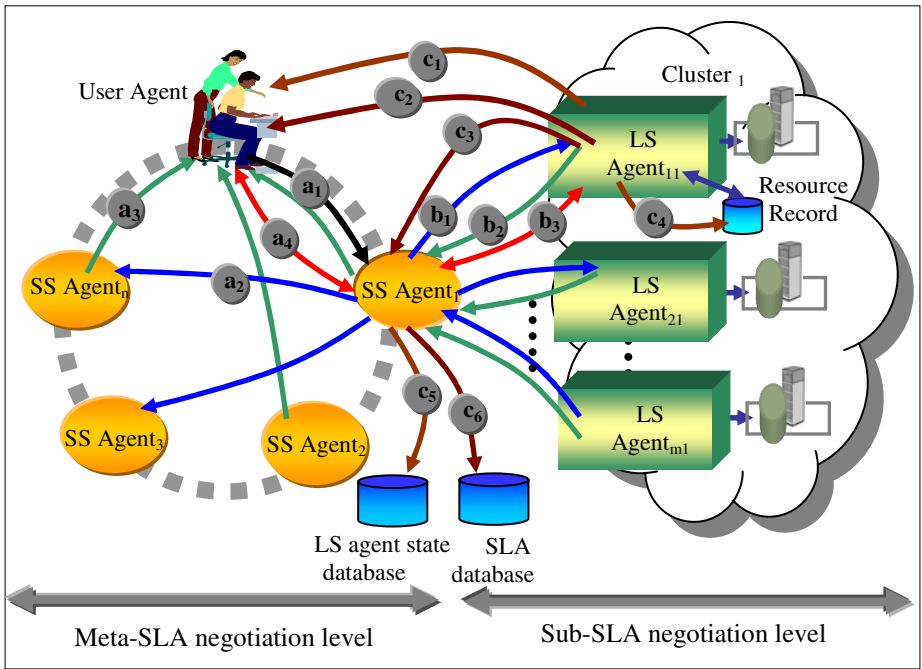


Fig. 2. Steps of the SLA negotiation protocol

**a. Meta-SLA negotiation steps:** Meta-SLA negotiation steps are the following:

**a<sub>1</sub>. Meta-SLA-request:** The user agent submits a meta-SLA to the nearest SS agent to request the execution of a job. The meta-SLA at this stage describes baseline user requirements such as required resources over a time period, job start time and end time, the cost the user is prepared to pay, required execution host information, etc.

**a<sub>2</sub>. Meta-SLA-announcement:** Upon receiving the meta-SLA request from the user agent, the SS agent advertises the meta-SLA to the neighbouring SS agents in the net by sending a meta-SLA-announcement.

**a<sub>3</sub>. Meta-SLA-bidding:** In response to the meta-SLA-announcement, each SS agent queries the LS agent state database to check the availability of required resources for the job response time to identify suitable set of resources. Cost is then evaluated for each potential resource. The SS agents select the best resources which satisfy the availability and cost constraints of the job. Each SS agent submits a meta-SLA-bid to the user agent. The meta-SLA-bid describes the new estimated response time and cost, and other additional information such as proposed compensation package.

**a<sub>4</sub>. Meta-SLA-award:** The user agent, upon receiving the meta-SLA-bids, selects the most appropriate SS agent with the best meta-SLA-bid and sends a notification agreement to its. The selected SS agent stores the agreed meta-SLA in the SLAs database. The user has now an agreement with the Grid provider to use its resources.

**b. Sub-SLA negotiation steps:** Once the SS agent has been awarded the agreement, it decomposes the meta-SLA by generating one or more sub-SLAs and negotiates the sub-SLAs with the local LS agents. The sub-SLA negotiation steps are the following:

**b<sub>1</sub>. Sub-SLA-announcement:** The SS agent sends the sub-SLA-announcements to the identified suitable LS agents within the cluster under its control.

**b<sub>2</sub>. Sub-SLA-bidding:** The LS agents, upon receiving the sub-SLA-announcements, query the resource record for information on CPU speed, memory size, etc. to identify suitable resources and submit bids to the SS agent.

**b<sub>3</sub>. Sub-SLA-award:** The SS agent evaluates the best sub-SLA in accordance to time and cost criteria and sends a notification agreement to the most appropriate LS agent with the best resources. The SS agent stores the sub-SLA in the SLAs database and updates the LS agent state information. The selected LS agents update the resource record and reserve the resources to execute the job.

**c. Task execution steps:** The basic stages for task execution are the following:

- c<sub>1</sub>.** The LS agent sends a notification of initiation of job execution to the user agent.
- c<sub>2</sub>.** At the end of job execution, a final report including the output file details is sent to the user agent.
- c<sub>3</sub>.** SL agent sends a notification end job execution to the SS agent.
- c<sub>4</sub>.** SL agent updates information held in the resource record database.
- c<sub>5</sub>.** SS agent updates the LS agent state database.
- c<sub>6</sub>.** SS agent updates the status of the SLAs in the SLAs database.

## 5 SLA Re-negotiation in the Presence of Uncertainties

Re-negotiation has been identified as a long-term goal of the RealityGrid UK e-Science Pilot Project [4]; here, simulations may be collaboratively visualized and steered as the simulation runs. Even once a SLA has been agreed, re-negotiation is required for multiple reasons: to extend the running time of an increasingly interesting simulation; to increase the computational resources dedicated to either the simulation, thereby accelerating the experiment, or to the visualization, in order to improve the resolution of the rendering; resources might also be given back when the improved speed or picture quality was no longer required. Also, more generally, resources may

fail unpredictably, high-priority jobs may be submitted, etc. In a busy Grid environment, SLAs would be constantly being added, altered or withdrawn, and hence scheduling would need to be a continual, dynamic and uncertain process. The introduction of re-negotiation, permitted during job execution (as well as before it commences) makes the schedule more dynamic, requiring more frequent rebuilding of the schedule.

In the presence of failures, the SS agents re-negotiate the SLAs in failure at the local and global levels in order to find alternative LS agents to execute the jobs in failure. The basic steps of failure recovery are the following, as shown in figure 3:

- f<sub>1</sub>**. LS agent<sub>11</sub> notifies SS agent<sub>1</sub> of the failure.
- f<sub>2</sub>**. SS agent<sub>1</sub> re-negotiates the sub-SLAs in failure to find alternative LS agents locally within the same cluster by initiating a sub-SLA negotiation session with the suitable LS agents.
- f<sub>3</sub>**. If it cannot manage to do so, the SS agent<sub>1</sub> re-negotiates the meta-SLAs with the neighbouring SS agents by initiating a meta-SLA negotiation session.
- f<sub>4</sub>**. SS agent<sub>2</sub>, ..., SS agent<sub>n</sub> re-negotiate the sub-SLAs in failure to find alternative LS agents.
- f<sub>5</sub>**. SS agent<sub>2</sub> located LS agent<sub>22</sub> to execute the job in failure.
- f<sub>6</sub>**. At the end of task execution, LS agent<sub>22</sub> sends a final report including the output file details to the user agent.
- f<sub>7</sub>**. In case the SS agent<sub>1</sub> could not find alternative LS agents at the local and global levels, the SS agent<sub>1</sub> sends an alert message to the user agent to inform him that the meta-SLA cannot be fulfilled and the penalty fee needs to be paid.

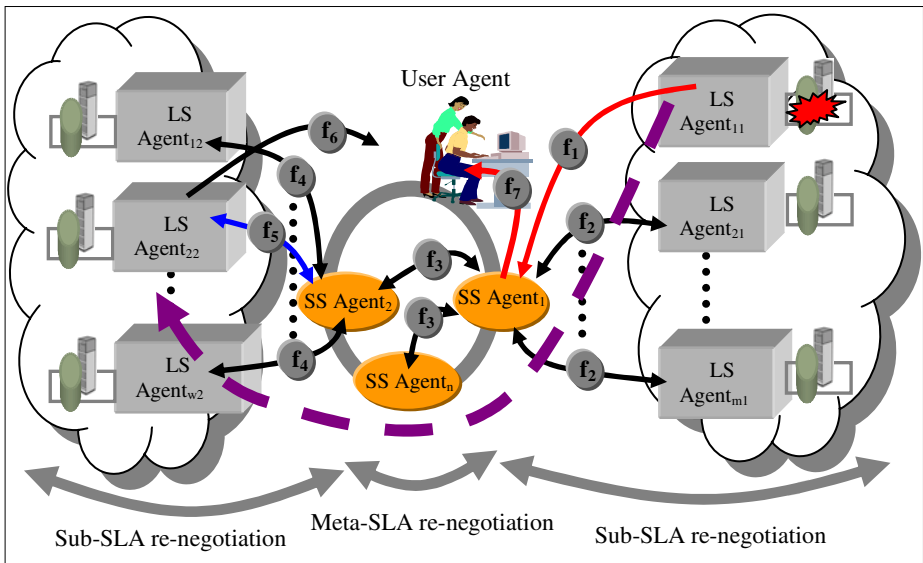


Fig. 3. Re-negotiation in the presence of resource failure

## 6 Conclusion

In this paper we have proposed a new infrastructure for efficient job scheduling on the Grid using multi-agent systems and a SLA negotiation protocol based on the Contract Net Protocol. Local autonomy and cooperation capabilities of the multi-agent architecture offer prospects for the most challenging issues in Grid scheduling which are: integration, heterogeneity, high flexibility, high robustness, reliability, and scalability. The SLA negotiation protocol proposed based on the Contract Net Protocol allows the user agent a flexible and robust negotiation of the execution of jobs on the Grid. The SLA negotiation protocol is a hierarchical bidding mechanism which yields many advantages relevant to Grid scheduling, such as dynamic task allocation, natural load-balancing as busy agents do not need to bid, dynamic introduction and removal of resources, and robustness against failures. To deal with the presence of uncertainties, re-negotiation is proposed to allow the agents to re-negotiate the SLAs in failure.

## References

1. Berman, F.: High performance schedulers. In: Foster, I., Kesselman, C. (Eds.): *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufman Publishers (1998) 279-307.
2. Cao, J., Jarvis, S.: ARMS: An agent-based resource management system for Grid computing. *Scientific Programming*, 10 (2002) 135-148.
3. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems. *Lecture Notes in Computer Science*, Vol. 2537 (2002) 153-183.
4. Czajkowski, K., Pickles, S., Pruyne, J., Sander, V.: Usage scenarios for a Grid resource allocation agreement protocol. *Draft Global Grid Forum Informational Document* (2003).
5. Ferber, J. (ed.): *Multi-agent systems: An introduction to Distributed Artificial Intelligence*. Addison-Wesley, London (1999).
6. Foster, I. and Kesselman, C. (eds.): *The Grid: Blueprint for a new computing infrastructure*. Morgan Kaufman Publishers (1998).
7. Frey, J., Tannenbaum, T., Livny, M.: Condor-G: a computational management agent for multi-institutional Grid. *Cluster Computing*, 5 (2002) 237-246.
8. GRAAP: GRAAP-WG, Grid resource allocation agreement protocol working group in the Global Grid Forum. Website: <https://forge.gridforum.org/projects/graap-wg/> (2004).
9. Hamscher, V., Schwiigelshohn, U., Streit, A., Yahyapour, R.: Evaluation of job scheduling strategies for Grid computing. *Lecture Notes in Computer Science* (2000) 191-202.
10. Keller, A., Kar, G., Ludwig, H., Dan, A., Hellerstein, J.L.: Managing dynamic services: A contract based approach to a conceptual architecture. *Proceedings of the 8<sup>th</sup> IEEE/IFIP Network Operations and Management Symposium* (2002) 513-528.
11. Krauter, K., Buyya, R., Maheswaran, M. A taxonomy and survey of Grid resource management systems. *Software Practices Experience*, 32 (2002) 135-164.
12. Ludwig, H., Keller, A., Dan, A., King, R.: A service level agreement language for dynamic electronic services. *Proceedings of the 4<sup>th</sup> IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems* (2002) 25-32.
13. MacLaren, J., Sakellariou, R., Garibaldi, J., Ouelhadj, D.: Towards service level agreement based scheduling on the Grid. *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services, in the 14<sup>th</sup> International Conference on Automated Planning & Scheduling*, Whistler, Canada (2004) 100-102.

14. O'Hare, G., Jennings, N. (Eds.): *Foundations of Distributed Artificial Intelligence*, Wiley, New York (1996).
15. Sakellariou, R., and Zhao, H.: A hybrid heuristic for DAG scheduling on heterogeneous systems. *Proceedings of the 13<sup>th</sup> International Heterogeneous Computing Workshop* (2004).
16. Shen, W., Norrie, D., Barthes, J. (eds.): *Multi-agent systems for concurrent intelligent design and manufacturing*, Taylor & Francis, London (2001).
17. Slowinski, R., and Hapke, M. (eds.): *Scheduling under fuzziness*. Physica Verlag (2000).
18. Smith, R.: The contract net protocol: high level communication and control in distributed problem solver. *IEEE Transactions on Computers*, 29 (1980) 1104-1113.
19. Zadeh, L.A.: Fuzzy Sets. *Information and Control*, 8 (1965) 338-353.