Forming and managing agent teams acting as resource brokers in the Grid — preliminary considerations

Wojciech Kuranowski
Software Development Department,
Wirtualna Polska
ul. Traugutta 115C, 80–226 Gdansk
wkuranowski@wp-sa.pl

Maria Ganzha, Maciej Gawinecki
Systems Research Institute,
Polish Academy of Science,
ul. Newelska 6, 01-447 Warszawa, Poland
(maria.ganzha,maciej.gawinecki)@ibspan.waw.pl

Marcin Paprzycki
Institute of Computer Science,
Warsaw School of Social Psychology,
ul. Chodakowska 19/31,
03–815 Warszawa, Poland,
Systems Research Institute, Polish Academy of Science,
ul. Newelska 6, 01-447 Warszawa, Poland
marcin.paprzycki@swps.edu.pl

Ivan Lirkov, Svetozar Margenov
Institute for Parallel Processing,
Bulgarian Academy of Sciences,
Acad. G. Bonchev, Bl. 25A,
1113 Sofia, Bulgaria
(ivan,margenov)@parallel.bas.bg

Abstract-

Recently, we can observe an increasing interest in utilization of software agents in computational Grids. In our work agent teams play role of Grid resource brokers and managers. Previously we have discussed how to efficiently implement matchmaking services, as well as proposed a way by which agents select a team that will execute their job. In this paper we focus our attention on processes involved in agents joining a team.

1 Introduction

Following an interesting trend claiming that software agents have an important role to play in Grid computing ([11, 18]) we have started investigating how teams of agents can be utilized as resource brokers and managers in the Grid. Specifically, in [10] we have presented an initial overview of the proposed approach. In [9] we followed with a study of the most effective way of implementing yellow-page based matchmaking services. Finally, in [8] we considered processes involved in agents seeking teams to execute their jobs. The aim of this paper is to start addressing the question: how agent teams are formed?

We start with an overview of the proposed system, beginning from the basic assumptions and following with an UML Use Case Diagram. In the next section we discuss issues involved in agent to agent-team matchmaking. Paper is completed with UML-based formalization of the main process involved in agent-team formation and agent team management and report on the status of the implementation. Work presented here is an extended version of [13].

2 System overview

Let us start by making it explicit that in our work we follow these researchers who claim that software agents will play an important role in design, implementation and long-term upkeep of large-scale software systems (see e.g. [16]). Second, more specifically in the context of this paper, our work explicitly assumes that there is an important role to be played by software agents in the future development of the Grid. While arguments to this effect can be found in [11, 18], in our view this assumption is further supported by the existing body of research devoted to combining software and the Grid and increasing number of conference papers which follow this path. Specifically, contributions of (1) J. Cao and colleagues, who combined PACE methodology with a hierarchical agent-based structure used for resource discovery [6], (2) B. Di Martino, O. Rana and collaborators, who have developed MAGDA (Mobile AGent Distributed Application) toolkit designed to support (1) resource discovery, (2) performance monitoring and load balancing, and (3) task execution within the grid [17], (3) S.S. Manvi and colleagues who suggested utilization of mobile agents which traverse the network to complete a user defined task [14] and (4) D. Ouelhadj and collaborators who studied negotiation (and re-negotiation) of a Service Level Agreement between agents representing resources and resource users [15] are worth mentioning as the most important thus far (for more details, see [8]). This being the case, we do not want to get involved in a discussion about importance or lack thereof of software agents (while admitting that there exist critics of this approach who claim, among others, that software agents are nothing more that active objects that are known since 1980th). In our work *we assume* that software agents do have a role to play in future of computing and investigate ways of utilizing them in context of Grid computing.

While in [8] we have summarized pros and cons of solutions proposed in the above mentioned key research results, here let us make explicit fundamental assumptions that underline our work (and are in a way a positive response to observed shortcomings of work done in the past).

1. The Grid is viewed as a global infrastructure; rather than a local / company / laboratory-based Grid and in this way we return to the most fundamental view of the Grid presented by Foster and Kesselman ([12]).

In the context of our work, the main difference is that in a "local" Grid nodes can be assumed to be under control of a "local administrator." However, in the case of a global Grid (consisting of, among others, home computers), the situation is similar to the P2P environment, where no centralized control over individual Grid nodes is exerted.

2. As a result, we have to take into account the fact that the global Grid understood this way is a highly dynamic environment, where node loads can change rapidly and, furthermore, nodes can disappear without much of a warning (e.g. envision a dog that runs in the room and disconnects the computer from the power outlet or the network, while its owner is at work and will notice this fact only about 8 hours later).

3. In the Grid, workers (in our case *agent workers*) that want to contribute their resources (and be paid for their usage), meet and interact with users (in our case *agent users*) that want to utilize (and are ready to pay for) offered services to complete their tasks.

4. From 2 and 3 follows that assuring Service Level Agreement (SLA) and fulfilling Quality of Service (QoS) conditions may be rather difficult (e.g. nobody can assure that a home PC that is executing users' job will be rebooted and job restarted when the machine goes down accidentally). While in the case of a project like SETI@HOME, this does not make much difference in which order results are collected and when a given specific result will be obtained, this is clearly no the case when money and deadlines are involved.

5. Agent mobility has to be treated with caution as it can solve problems of network congestion (rather than generating them) only when decisions: what to move, when and where are part of system design and the total agent mobility minimized. If an agent carrying a code and data has to make multiple jumps before it finds the place where the job will be executed, then the price may be very high (depending directly on the number of hops and size of the load) making this approach much worse that its static counterpart.

As a result of these assumptions in [10] we have proposed a system based on the following tenets:

- agents work in teams (groups of agents)
- each teams has a single leader—*LMaster agent*
- each *LMaster agent* has a mirror *LMirror agent* that can take over its job in case when it "goes down"
- incoming workers (*worker agents*) join agent teams based on individual set of criteria (which can change over time)
- teams (represented by their *LMaster*s) accept workers based on individual set of criteria (which can change over time)
- decisions about joining and accepting involves multicriterial analysis
- each *worker agent* can (if needed) play role of an *LMaster* or an *LMirror*
- matchmaking is provided through yellow pages [19] and facilitated by the *CIC agent* [4]

Combining these propositions resulted in the system represented in Figure 1 as a Use Case diagram.

Let us now focus our attention on interactions between the *User* and its representative: the *LAgent* and agent teams existing in the system (additional discussion can be found in [10, 8]). Let us assume that the system is already "running for some time", so that at least some agent teams have been formed and collected data about their interactions with *User*s of the system. Similarly, *User*s of the system had a chance to collect data about using various teams to execute their jobs and about being a member of various teams. Furthermore, team "advertisements" describing: (1) what resources they offer for prospective users, and (2) characteristics of workers they would like to see joining their team, are posted with the *Client Information Center* (*CIC*) (an extensive discussion about the role of the *CIC* and its implementation can be found in [9]. Let us note that the *User* depicted in Figure 1, can either contribute resources to the Grid, or utilize resources available there and these roles can change. Specifically, one day the *User* may find in the Grid (within one of agent teams) Maple software that it needs, but does not have, while another day she may contribute and be paid for the raw computational power of her multi-core PC. Interestingly, both situations are "Use Case symmetric" and involve the same pattern of interactions between agents representing the *User* and agent teams in the system.

User who wants to utilize resources available in the Grid communicates with its local agent (*LAgent*) and formulates conditions for executing a job (e.g. the computational resources needed and the price she is willing to pay). The *LAgent* communicates with the *CIC* to obtain a list of agent teams that satisfy its predefined criteria (e.g. have the right hardware and software configurations). After obtaining such a lit, the *LAgent* eliminates from it teams that are not deemed trustworthy (e.g. teams that broke the SLA in the past; see [5] for more details about trust management in a similar system). If no team is trustworthy enough, the *LAgent* communicates this to its *User* and awaits further instructions. If there were teams deemed trustworthy, the *LAgent* communicates with their representatives (*LMasters*). This communication takes form of the FIPA Contract Net Protocol [1]. First, the *LAget* communicates what job it needs executing and under what conditions; second, it obtains job execution proposals; and third, it evaluates them using multicriterial analysis. If the *LAgent* finds a team that it would like to execute its job, a contract is formed. If no such team is found (e.g. if nobody is willing to execute a

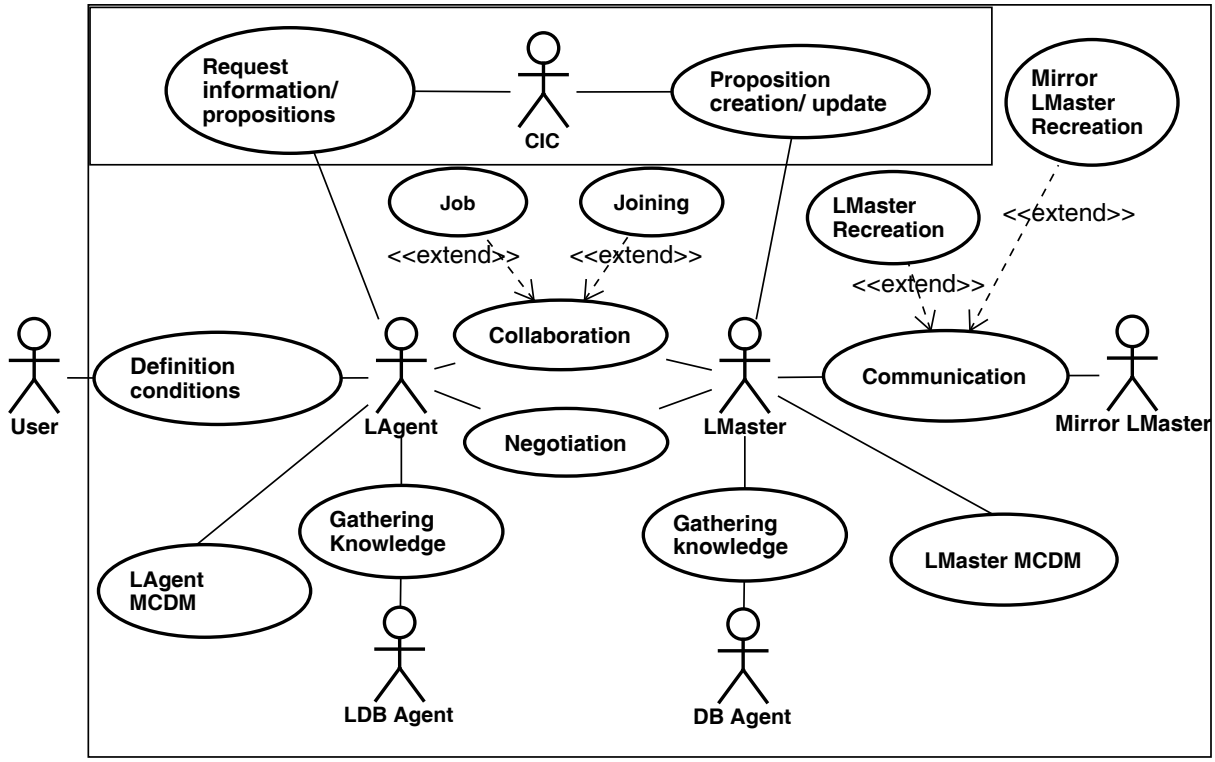

Figure 1: Use Case diagram of the proposed system

10 hour job for 5 cents), the *LAgent* informs its *User* and awaits further instructions. For a complete description of this process, see [8].

The remaining part of this paper will be devoted to the situation when *User* requests that its *LAgent* joins a team and work within it (e.g. to earn extra income for the *User*).

3 Selecting team to join

The overall schema of interactions involved in an *LAgent* selecting a team to join is very similar to that described above for an agent selecting a team to execute its job. First, the *User* specifies the conditions of joining, e.g. minimum payment for job execution, times of availability etc. Then she provides its *LAgent* with the description of resources offered as a service, e.g. processor power, memory, disk space etc. The *LAgent* queries the *CIC* which agent teams seek workers with specified characteristics. Upon receiving the list of such teams, it prunes teams deemed untrustworthy (e.g. teams that did not deliver on promised payment, or that never send a job to be executed by a given *LAgent*) and contacts *LMasters* of the remaining teams (if no team is left on the list, the *LAgent* informs its *User* and awaits further instructions). Again, negotiations between the *LAgent* and the *LMasters* take form of the FIPA Contract Net Protocol. The summary of this process is depicted as a sequence diagram in Figure 2.

For clarity, this sequence diagram is simplified and does not include possible negative responses and/or errors that can take place during message exchanges. Observe also that registering with the *CIC* takes place only once — when a new *LAgent* joins the system. All subsequent interactions between the *CIC* and a given *LAgent* involve only checking credentials (making sure that a given agent is registered with the system). The sequence diagram includes also processes involved in "mirroring." As stated in Section 2, in our system we assume that the *LMaster* has its mirror, the *LMirror* agent. The role of this agent is to become the *LMaster* in the case when the current *LMaster* "disappears." To understand this design, let note that it is only the *LMaster* that has complete information about team members, jobs that are executed (and by which team member), etc. Therefore, sudden disappearance of the *LMaster*, for all practical purposes, would "destroy the team" as all this vital information would be lost. Furthermore, worker agents would have no way to communicate results of their work (as they would not know who is the job owner), etc. To avoid such a situation the *LMaster* shares all necessary information with the *LMirror*, who will take over immediately if it recognizes that the *LMaster* is "gone," and as its first order of business will promote one of *worker agents* to become the new *LMaster*. Note that in the case when the *LMirror* agent "goes down," the *LMaster* will immediatly promote one of *agent workers* to become the new *LMirror*. Obviously, it is possible that both the *LMaster* and the *LMirror* "fail" simultaneously and the team will be dissolved. However, our goal is not to build a fault tolerant environment, but only to introduce some degree of resilience (for a reasonable price). However, since

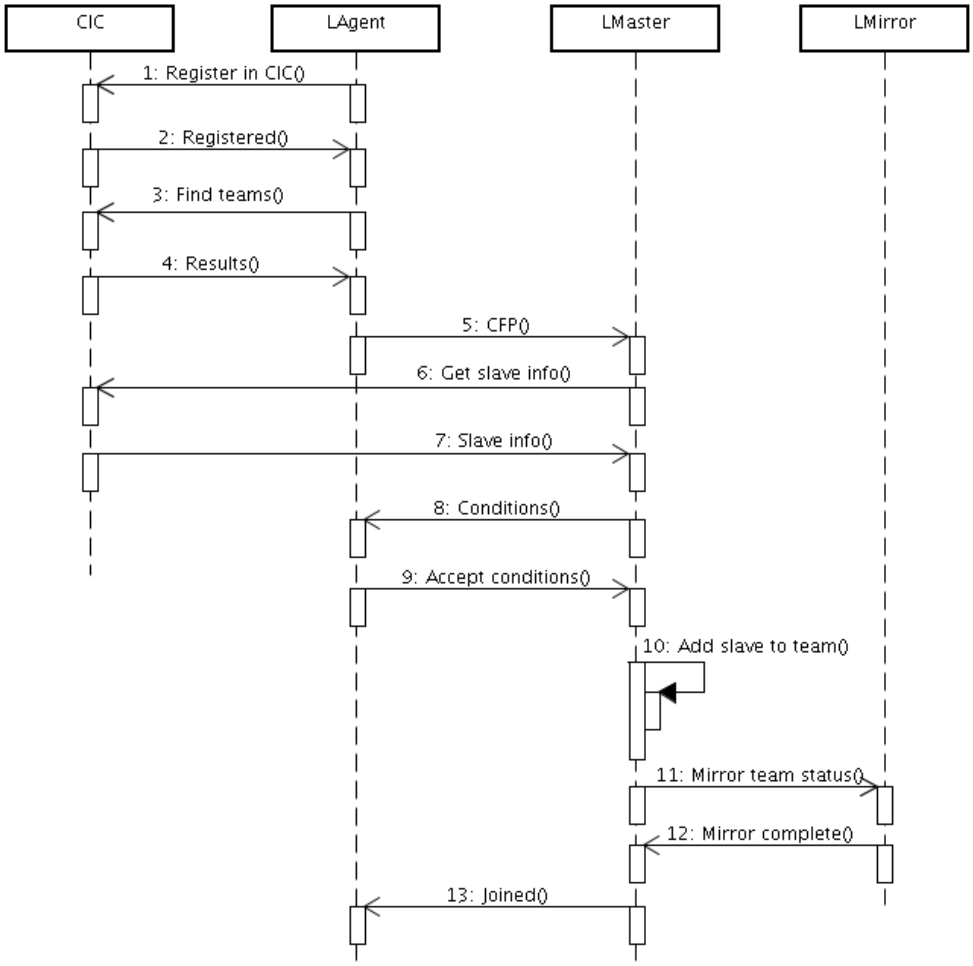

Figure 2: Sequence diagram of interactions when an agent is seeking a team to join.

this interesting in its own right subject is out of scope of this paper it is omitted from further considerations.

3.1 Representing conditions of joining

Let us now discuss representation of resources that the *LAgent* brings to the team and its conditions of joining. Based on current prevailing trends, we have decided to extensively utilize ontologies (semantical data demarcation) in our system. Since we want to use our agents in the Grid, an ideal situation would be if an all-agreed "ontology of the Grid" (that would include both specification of resources and of the economical model) would exist. We would then simply utilize it directly in our agent system. Unfortunately, while there exist separate and incompatible attempts at designing such an ontology (some of them focused on a particular Grid middleware, while others on a domain of Grid application), currently they can only be considered as "work in progress" toward the common ontology of the Grid. Therefore, instead of selecting one of them and paying the price of dealing with a large and not necessarily fitting our needs ontology (which would in turn mean that we would have to make changes in an ontology that we have not conceived and have no control over), we focus our work on the agent-related aspects of the system (designing and implementing agent system skeleton) while utilizing simplistic ontologies; and readers should keep this fact in mind. Obviously, when a common Grid ontology will be agreed on, our system *will be ready* for it and adaptation will be rather simple. Currently, our ontology of Grid resources is focused on their "computational" aspects, e.g. processor, memory and available disk space. What follows is a snippet of our OWL Lite based ontology:

```
:Computer
        :a owl:Class .

:hasCPU
        :a owl:ObjectProperty ;
        rdfs:range :CPU;
        rdfs:domain :Computer.

:CPU
        :a owl:Class .

:hasCPUFrequency
        :a owl:DataProperty ;
        rdfs:comment "in GHz";
        rdfs:range xsd:float ;
```

```
        rdfs:domain :CPU.

:hasCPUType
        :a owl:ObjectProperty;
        rdfs:range :CPUType;
        rdfs:domain :CPU.

:CPUType
        :a owl:Class.

Intel :a :CPUType.
AMDAthlon :a :CPUType.

:hasMemory
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.


:hasUserDiskQuota
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.

:LMaster
  :a owl:Class;

:hasContactAID
        :a owl:ObjectProperty;
        rdfs:range xsd:string;
        rdfs:domain :LMaster.

:hasUserDiskQuota
        :a owl:DatatypeProperty;
        rdfs:comment "in MB";
        rdfs:range xsd:float;
        rdfs:domain :Computer.
```

Let us now assume that worker *PC1425* which has a 3.2 GHz Intel processor, 1025 Mbytes of memory and 2000 Mbytes of disk space available as a "Grid service." In our ontology it would be represented as:

```
:PC1425
        :a :Computer;
        :hasCPU
        [
                a :CPU;
                :hasCPUType :Intel;
                :hasCPUFrequency "3.2";
        ];
        :hasUserDiskQuota "2000";
        :hasMemory "1024".
```

Observe that resource describing information is used in two situations. First, each team looking for members advertises specific resources it is looking for. Such an advertisement is an instance of an ontology. However, in the case of seeking team members we have to assume that parameters with numerical values (e.g. processor speed or available disk space) have to be treated as minimal requirements. In other words, asking for team members with processor speed of 2.5 GHz does not exclude computers with processors running at 2.7 GHz. At the same time, alphanumeric parameters that describe, for instance, necessary software have to be treated as hard constraints that have to be satisfied (if a team is seeking members that own MATLAB, then only such computers are of interest). Note also that descriptions of sought workers include only resource describing parameters, but they do not include specific offers related to, for instance, payments for working for the team, as these are subject to negotiations.

Second, when the *LAgent* requests list of teams that look for members, information about its own resources is used as a filter in the query. In our work, we use Jena [2] to store ontologically demarcated information, and SPARQL query language [3]. Therefore, when the *LAgent* representing the above described computer communicated with the *CIC*, the following SPARQL query would be executed.

```
PREFIX Grid: <http://Gridagents.sourceforge.net/Grid#>
SELECT ?team
WHERE {
  ?team Grid:needs ?machine .
  ?machine Grid:hasCPU ?cpu ;
           Grid:hasMemory ?mem ;
           Grid:hasQuota ?quota .
  FILTER ( ?cpu <= "3.2"^xsd:float ) .
  FILTER ( ?mem <= "1024"^xsd:integer ) .
  FILTER ( ?quota <= "20000"^xsd:integer ) .
}
```

Obviously, our decision of utilizing ontologies in the system extends beyond computational resources and includes also description of various "conditions" that are imposed on various "actions." This is the most natural way of providing semantic support for agent-agent negotiations. In [8] we have presented the way that we represent conditions of selecting team to execute a job. In the case of conditions of joining a team we are currently utilizing only three parameters: (1) price per work-hour (requested by the *Lagent* or offered by the *LMaster*), (2) work time—specific times of the day when the resource is to be available (may include non-stop availability), and (3) length of contract—time interval that a given *LAgent* is offering to be a member of a given team. In the latter case we assume that it is realistic that a given *LAgent* joins a given team for a specific limited time. Obviously, while a contract holds for such a limited time, if both sides are satisfied, it can be extended for subsequent (and possibly longer) time periods. While such a contract extension could be conceptualized in a special form of negotiations (that would involve only a given *LAgent* and a given *LMaster*), this does not have to be so. Note that we assume that our system will employ trust management procedures. This being the case, very high level of trust between the *LAgent* and a given team will naturally influence their choice of continuing working together. Finally, note that again, we are using a very simplistic set of conditions. However, extending this set to a more robust one is purely a technical issue that does not affect the proposed approach.

Let us now present an instance of a Call for Proposals (the first step of the FIPA Contract Net) that is send by the *LAgent* and that consists of two parts (depicted in Figure 3.1). First, the description of the same computer as above; this time being offered to become a member of a team: (1)

```
(cfp
  :sender (agent-identifier :name proteus@bach:1099/JADE)
  :receiver (agent-identifier :name zerg@chopin:1099/JADE)
  :content
    ((action
      (agent-identifier :name zerg@chopin:1099/JADE)
      (take-me
        :configuration (hardware
          :cpu 3.2
          :memory 1024
          :quota 2000)
        :conditions (condition
          :availability (frequency
            :unit (day)
            :day-time (period
              :from 00000000T23500000
              :to 00000000T08150000))
          :contract-duration +00000007T000000000))
  :language fipa-sl0
  :ontology joining-ontology
  :protocol fipa-contract-net
)
```

Figure 3: Call for proposals utilizing FIPAS SL and ontology of resources and conditions of joining.

with an Intel processor running at 3.2 GHz, (2) that offers to users 1024 Mbytes of RAM and (3) 2 Gbytes of disk space. Second, the specific conditions that are imposed by its *User*: (4) it is to be available every night between 23:50 and 8:15, and (5) the suggested length of contract is to be 7 days. Note that payment conditions are not specified; they are private to the *LAgent*. Since we use the FIPA Contract Net Protocol, we assume that the proposed payment is a part of the response of the *LMaster*. The message is utilizing the FIPA Semantic Language to specify that it is a FIPA Contract Net *proposal* message. This message utilizes ontologically demarcated information about the computer and about conditions of joining. In this way we are combining two ontological demarcations. One on the level of agent messaging and one representing the semantics of the Grid.

Separately, let us note that each time a given *LAgent* issues a CFP it may specify different resource as: (1) the same *LAgent* may represent *User*'s different machines, or (2) for a single machine at one time available disk space may be 5 Gbytes, while at another time 25 Gbytes (e.g. depending on the number and size of JPEG files just uploaded from the camera).

3.2 Negotiations

Let us now focus our attention on negotiations. The first step is the *LAgent* sending a CFP (arrow 5 in Figure 2) containing resource description and conditions of joining (see Figure 3.1) to all *LMaster agent*s remaining on the list of contact points of trusted teams. Upon receiving the CFP each *LMaster* contacts the *CIC* to make sure that this particular *LAgent* is registered with the system (arrows 6 and 7 in Figure 2). This step has been added to somewhat improve the overall security of the system. We have simply assumed that only *LAgent*s that are registered with the *CIC* can join agent teams (or have their jobs executed by teams existing in the system). In other words, in addition to providing matchmaking services, the *CIC* keeps track of all agents registered with the system and only such agents are allowed to participate in it. Let us note that the apparent single point of failure of the system (the *CIC* agent) is in reality only a technical problem. To understand this point observe that, for instance, all *gmail* users log in into *gmail* through a single address: http://www.gmail.com/ and it does not matter how the *gmail* service is actually implemented. This solution is our metaphor behind the *CIC* and services it provides.

On the basis of the CFP, *LMaster*s prepare their response. First, CFPs that do not satisfy hardware / software requirement are refused (e.g. worker that does not have Maple, cannot join a team that requires Maple). Let us note that this step is necessary not only because some *LAgent*s may send incorrect offers in their CFP. What we are more concerned with is the following scenario (resulting from a completely asynchronous nature of the system). Let us assume that a given team $X25$ decided to change its conditions of joining. While it send message to this effect to the *CIC* an *LAgent* executed a query that returned team $X25$ within the list of candidates, still based on its old joining conditions. It then submitted a CFP to team $X25$, which did not match its current conditions of joining. Obviously, such offers have to be rejected immediately. Second, for each acceptable CFP, each *LMaster* prepares an offer to be send back. While there are multiple ways of doing so, we propose the following relatively simple one. The *LMaster* utilizes its knowledge about past jobs to establish a *base price*

per hour: B_c and a *base system* that matches it. For example, it can establish that a system like the one that appears in our example will be pail 5 cents per hour of utilization. Let us note that, due to the way that our ontology of resources has been developed, this price is split between three components (processor speed: P_b, memory: M_b, disk space: D_b). Obviously, in a case of a more complicated ontology of resources, a similar formula that includes all of them would be created. As a result we obtain what can be named as: processor cost P_c, memory cost M_c and disk cost D_c; specified in such a way that the base cost $B_c = P_c + M_c + D_c$. This information is then used to estimate the "value" of the new potential worker in the following way, (let us assume that the potential worker has processors speed P, memory M and disk space D):

$$Cost = \alpha\Big(\frac{P}{P_b}P_c + \frac{M}{M_b}B_c + \frac{D}{D_b}D_c\Big), \qquad (1)$$

Where $\alpha \in [0, 1]$ denotes the overhead charged by the *LMaster*. For instance $\alpha = 0.9$ means that the *LAgent* will be offered 90% of its value calculatd on the basis of its three components, while 10% will be collected by the *LMaster*.

Obviously, this model is extremely simplistic, but our goal was not to build a complete economical model of the Grid (for this, one would need a Grid ontology to start with). Note, however that replacing it with a more complicated one requires just substitution of a single module in the *LMaster*.

Responses from *LMaster*s can have the following forms: (1) rejection (an ACL-REFUSE message) — in the case when the initial offer was incorrect, (2) lack of response in a predefined by the *LAgent* time — which could represent a connectivity problem, or the fact that the *LMaster* went down before the information about the incoming CFP was forwarded to the *LMirror*, (3) a specific offer (an ACL-RESPONSE message) — in our simplistic model such an offer would contain only the *price* offered in the case of joining the team.

The *LAgent* awaits for a specific time for responses from *LMasters* and then evaluates them. Since currently the response contains only the price, there are two possibilities. First, neither of proposals contains an offer that is higher than the user specified minimal price. In this case the *User* is informed and the *LAgent* awaits further instructions (this allows the *User* to "manually" accept one of the proposals even if they did not match her original requirements). Second, at least one of the offers is above *LAgent*'s reservation price, an agent teams is selected to be joined (arrow 9 in Figure 2). Note that in the case when two offers are the same, trust information can be naturally used to evaluate them and the team with a higher trust score wins the contract. Let us also note that as soon as a more complicated response is to be used (e.g. response that would contain tiered pricing and a guarantee of a certain number of contracts within the timespan of the contract) a multicriterial analysis would have to be applied [7].

Finally, observe that the final (re)confirmation is depicted as arrow number 11 in Figure 2. According to the Contract Net Protocol, since the *LAgent* was the originator of the negotiations, it has to be the receiver of the final confirmation that closes the protocol

4 Implementation

Currently we are in the process of implementing the above described processes. However, to be able to implement agents joining the team, we have to implement also additional mechanisms involved in agent team management. To illustrate the state of our implementation, in Figure 4, we present the GUI of an *LMaster* agent. Obviously, this GUI is presented only for illustration purposes (and used for testing the system), as we assume that each *LMaster* will run autonomously, with *Agent-User* interaction taking place only in a few, above described, conditions. In the context of agents joining teams, the most important informations, are (1) the *Workers requirements* box and (2) the *My Workers* box. The first one specifies that this *LMaster* is interested in workers that have (at least) 1 processor with speed between 1.0 and 3.0 GHz, memory of 1-4 Gbytes and that have disk space of (at least) 5 Gbytes available as a service. At the same time we can see that this *LMaster* is currently managing a team of 5 workers.

4.1 Managing the team

Let us now discuss in some detail the remaining functions that have been implemented and are related to team management. Here, the most important function that we started our work from is the "liveness" of the team. In other words, we want to give the *LMaster* ability to make sure that its team members are still alive (which should mean that they are executing their jobs—see below). Our current approach is summarized within the *Other configuration* box.

To assess the state of each team member, *LMaster* is performing testing functions in rounds (monitoring sessions). Each monitoring session consists of a certain number of tests (parameter *Number of tests*; in our case 10), while each test consist of a certain number of pings (parameter *Pings per test*; in our case 15). Therefore, in the case depicted in Figure 4, 10 tests of 15 pings would be send to each of 5 team members. Pings are send in an interval (parameter *Ping interval*; in our case 500 milliseconds). A given ping is counted as a success if a response comes within predefined time (parameter *Max ping reply*; in our case 300 milliseconds). Pings are send in a round-robin fashion and a ping to the next agent is send only when processing the ping to the given agent is completed. Failed pings are counted (as percent of failures) against the total number of pings in a single test. At the end of each test a score is produced for each agent and an agent fails a test if its percent failure is higher than the *Max loss* parameter (in our case loss higher than 50% would result in failing the test). We have also specified the number of tests that a given *LAgent* has to pass (parameter *Tests to pass*; in our case, with 8 tests in a round) to be considered to be a "live" worker agent. Our system allows that the *LMaster* automatically removaes failing *LAgent*s from the team. In other words, *worker agent*s that have been recognized as non-responsive in a given monitoring session will be removed from the team (this is the

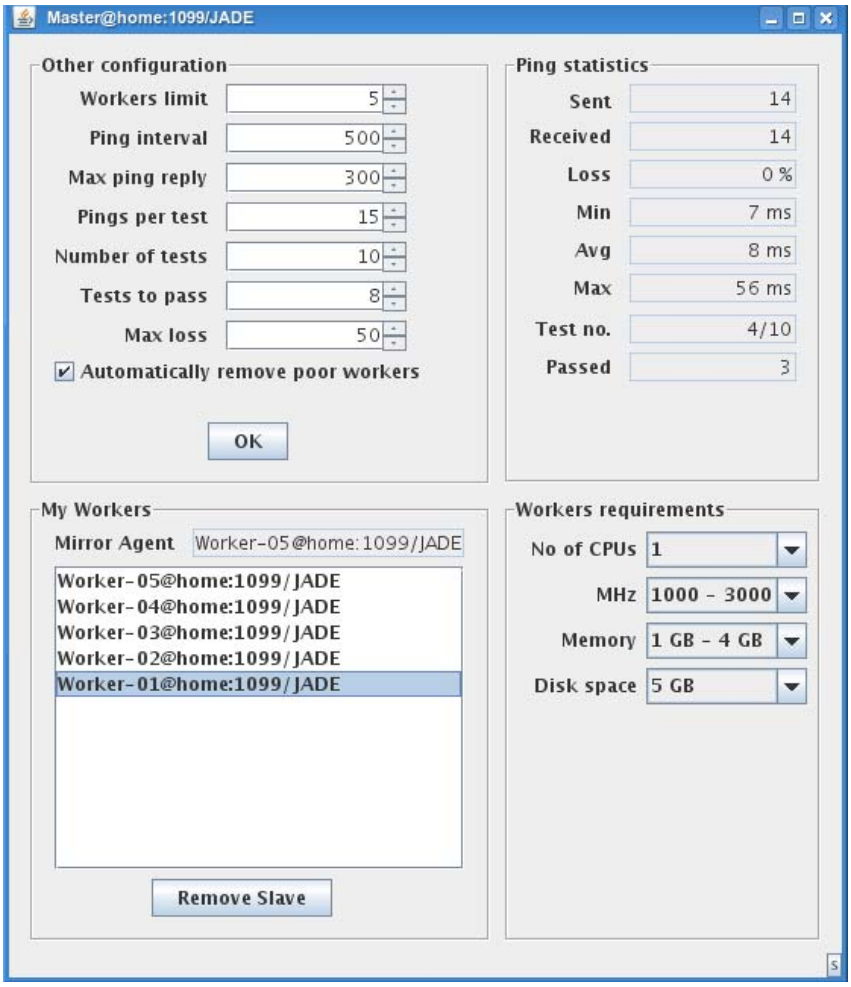

Figure 4: GUI of the *LMaster* agent.

case in Figure 4 since an apprpriate box is checked). After a completed monitoring session, all counters are zeroed and a new such session starts.

Finally, in the *Other configuration* box we can also see there that this *LMaster* will accept no more that 5 workers (which means it has a complete set of workers and should de-list its advertisement that it is seeking workers from the *CIC*). Let us also observe the *Ping statistics* box which provides statistical results of a current monitoring sessions.

All these functions have been implemented and tested for variety of parameter values. However, what is missing is a more involved logic that would answer at least two questions: (1) when an agent should actually be removed from a team? (is really a single failure within a monitoring round enough to disqualify a team member?), (2) what to do in the case of removing an agent from a team (how to deal with the job that a given agent was responsible for, what if this agent comes back and delivers expected results on time, what if it delivers results too late? etc.). Let us also note that the proposed mechanism is only providing us with information that a given agent is "alive" (as long as it is willing to respond to our pings) but tells us nothing if it actually executes a job. Furthermore, since *LAgent*s can "lie" about what they are doing, dealing with this issue will involve, among others, trust considerations and has to be considered within a big picture of trust considerations involved in utilization of agent teams in the Grid. We plan to address this issue (truss management) in a comprehensive way in the near future.

5 Concluding remarks

The aim of this paper was to introduce basics of agent team formation and management, within the framework of the earlier proposed agent-team-based Grid resource brokering and management system. We have presented a complete description of processes involved in agent joining the team, which while relatively simplistic at this stage, can be easily augmented to a more robust version. Currently we are proceeding with implementation of the above described processes. This involves also development of agent team management tools that have been presented in Section 4. Obviously, there exists a number of research issues that have to be addressed and at least some of them have been outlined above. Our first goal will be to conceptualize trust management in the proposed system and we plan to report on this issue in the near future.

Acknowledgments

Work presented here is a part of the Poland-Bulgaria collaborative grant: "Parallel and distributed computing practices".

Bibliography

[1] Fipa contract net protocol specification. http://doi.www.fipa.org/specs/fipa00029/SC00029H.html.

[2] Jena—a semantic framework for java. http://jena.sourceforge.net.

[3] Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query.

[4] C. Bádicá, A. Báditá, M. Ganzha, and M. Paprzycki. Developing a model agent-based e-commerce system. In J. L. et. al., editor, *E-Service Intelligence - Methodologies, Technologies and Applications*. Springer. in press.

[5] C. Bádicá, M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki. Towards trust management in an agent-based e-commerce system - initial considerations. In A. Zgrzywa, editor, *Proceedings of the MISSI 2006 Conference*, pages 225–236. Wroclaw University of Technlogy Press, Wroclaw, Poland.

[6] J. Cao, D. J. Kerbyson, and G. R. Nudd. Use of agent-based service discovery for resource management in metacomputing environment. In *Euro-Par '01: Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, pages 882–886, London, UK, 2001. Springer-Verlag.

[7] J. Dodgson, M. Spackman, A. Pearman, and L. Phillips. *DTLR multi-criteria analysis manual*. UK: National Economic Research Associates, 2001.

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

[9] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 327–335. PTI Press.

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

[11] I. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 8–15, Los Alamitos, CA, 2004. IEEE CS Press.

[12] I. Foster and C. Kesselman. The grid 2: Blueprint for a new computing infrastructure. 2003.

[13] W. Kuranowski, M. Paprzycki, M. Ganzha, M. Gawinecki, I. Lirkov, and S. Margenov. Agents as resource brokers in grids—forming agent teams. In *Proceedings of the 6th conference Large-Scale Scientific Computations*, London, UK, 2007. Springer-Verlag.

[14] S. Manvi, Birje, and B. Prasad. An agent-based resource allocation model for computational grids. *Multiagent and Grid Systems*, 1(1):17–27, 2005.

[15] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, K. Krishnakumar, and A. Meisels. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In *Advances in Grid Computing - EGC 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 651–660, Germany, 2005. Springer Verlag.

[16] J. N. R. An agent-based approach for building complex software systems. 44.

[17] O. F. Rana and B. D. Martino. Grid performance and resource management using mobile agents. *Performance analysis and grid computing*, pages 251–263, 2004.

[18] H. Tianfield and R. Unland. Towards self-organization in multi-agent systems and grid computing. *Multiagent and Grid Systems*, 1(2):89–95, 2005.

[19] D. Trastour, C. Bartolini, and C. Preist. Semantic web support for the business-to-business e-commerce lifecycle. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 89–98, New York, NY, USA, 2002. ACM Press. http://acm.org/10.1145/511446.511458.