Selecting grid-agent-team to execute user-job — initial solution

Mateusz Dominiak
Technical University of Warsaw, Department of Mathematics and Information Sciences
Warsaw, Poland
mateusz.dominiak@gmail.com

Maria Ganzha, Marcin Paprzycki
Systems Research Institute Polish Academy of Science
ul. Newelska 6, Warsaw, Poland
{maria.ganzha, marcin.paprzycki}@ibspan.waw.pl

Abstract

Recently we have proposed a novel approach to utilizing agent teams as resource brokers and managers in the Grid. Thus far we have presented an overview of the proposed approach discussed how to efficiently implement the information center, where agent teams advertise their needs and resources. In this paper we focus our attention on the way that user selects agent team that will execute its job. Details of initial implementation are presented and discussed.

1. Introduction

Grid computing is emerging as a promising approach to utilizing heterogeneous, geographically distributed computer resources. It is expected that virtualization of computing resources by Grid computing will facilitate creation of a new computing infrastructure consisting of readily available, adaptable resources. As a result, a broad impact on science, businesses and industries is expected [11]. Unfortunately, the uptake of the Grid, while speeding-up recently, is still unsatisfactory. One possible reason for this situation is an overly complicated support for resource management provided by current Grid software infrastructure.

In this context it has been suggested that software agents combined with semantical demarcation of resources may provide the necessary infrastructure, by infusing the Grid with intelligence [10, 15]. While this claim is not without critics (e.g. there are those who claim that service oriented architecture is all that is needed to solve exactly the same set of problems as software agents are supposed to [5]) in our work we accept arguments presented in [10, 15] as the starting point of our work. Therefore, we have searched for the existing solutions that would put software agents into the Grid.

The initial work on agents in grids can be traced at least to [6], where J. Cao and colleagues combined PACE methodology with a hierarchical agent-based structure used for resource discovery. While interesting, this work seems to be a road to nowhere as the PACE infrastructure, seems to be extinct. More recently B. Di Martino and O. Rana have proposed MAGDA (Mobile AGent Distributed Application), a mobile agent toolkit designed to support (1) resource discovery, (2) performance monitoring and load balancing, and (3) task execution within the grid [14]. While based on collaborating agents, the proposed system does not have an economic model associated with it. From our perspective, this is a substantial drawback, as the model of the Grid that we are interested in is such in which resource providers put their resources (e.g. computers) on the Grid, to be remunerated for their usage [4].

Starting form the economic model, S.S. Manvi and colleagues suggested utilization of (single) mobile agents which traverse the network to complete a user defined task [12]. While visiting nodes they establish local conditions for job execution. If these conditions (involving economic considerations) are acceptable, agents execute their job there (otherwise, they move on). Unfortunately, usage of a single agent makes the proposed approach highly vulnerable to adverse events, such as a node "disappearing without a trace" while the agent is executing its job there.

Also last year, D. Ouelhadj and colleagues considered negotiation (and re-negotiation) of a Service Level Agreement between agents representing resources and

resource users [13]. Negotiations were to be based on Contract Net Protocol and were focused on higher level functionalities of the system. As it will be seen, we follow the idea of utilizing Contract Net Protocol in our proposed approach.

Summarizing, the proposed approaches:

1. were somewhat limited in scope and functionality,
2. did not involve economical foundations/models,
3. relied on agent mobility, while not considering its cost—since agents carry tasks, their size depends on the size of transported code and data and thus agent mobility should be used very judiciously,
4. did not take into account full effect of grids highly dynamic nature and used single service providers—leaving users vulnerable to potential rapid fluctuations of workload of individual nodes, as well as nodes disappearing and reappearing practically without warning,
5. did not provide methods for trust management, which seems necessary when one takes into account expected reliance on "barely known" service providers.

As a way to address these issues we have presented a conceptual framework for an *agent team*-based approach to resource brokering in the Grid [9]. In a follow-up work, [8] we have discussed the way in which the central, yellow-page information carrying service can be efficiently implemented to prevent it from becoming a bottleneck of the system. The aim of this paper is to discuss how the agent representing its user can autonomously pick the team that will execute its task.

We proceed as follows, in the next section we summarize the design of our system. We follow with the description of the basic ontology that underlies the task execution negotiations. A detailed description of the way we have implemented it completes our work.

2. System description

In our work we view the Grid as an environment in which 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 offered services to complete their tasks. Obviously, *agent workers* can become *agent users* and vice-versa. In [9] we have proposed a system based on the following assumptions:

- agents work in teams (groups of agents)
- each teams has a single leader—*LMaster agent*
- each *LMaster* has a mirror *LMirror agent* that can take over its job in case when it "goes down"
- incoming workers (*worker agents*) join teams based on individual set of criteria
- teams (represented by their *LMaster*s) accept workers based on individual set of criteria
- decisions about joining and accepting involves multicriterial analysis (performed by so-called *MCDM modules*)
- each *worker agent* can (if needed) play role of an *LMaster*
- matchmaking is provided through yellow pages [16] and facilitated by the *CIC agent* [2]

Combining these assumptions 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: *LAgent* and agent teams residing in the system (information about the remaining parts of the system can be found in [9]). To do this let us assume that the system is already "running for some time," so that at least some agent teams have been already formed. As a result, team "advertisements" describing: (1) what resources they offer and/or what jobs they would like to execute, and (2) what "types" of agents they would like to join their team are posted with the Client Information Center *CIC*. As it was discussed in [8], currently we use a single *CIC* and since it is implemented on a single computer, we use a threaded-solution to maximize obtained throughput. At the same time we do recognize that this solution may become a bottleneck in the system and in the future we may utilize an approach similar to that reported in [6].

Let us note that the *User*, represented in Figure 1, can be either someone who tries to contribute resources to the grid, or someone who would like to utilize resources available there. Interestingly, the Use Case diagram shows that both situations are "symmetric" and involve the same pattern of interactions between the *User* and agents representing her and the system (moreover processes that take place here are very similar to these described in [2] and interested user may consult it for further details). Let us start our description from the case of *"User-contributor"* who wants to contribute resources to the grid. She communicates with her agent (the local agent *LAgent* which becomes a worker agent) and formulates conditions for joining an agent team (or requests that a new team be created). The *LAgent* communicates with the Client Information Center *CIC* to

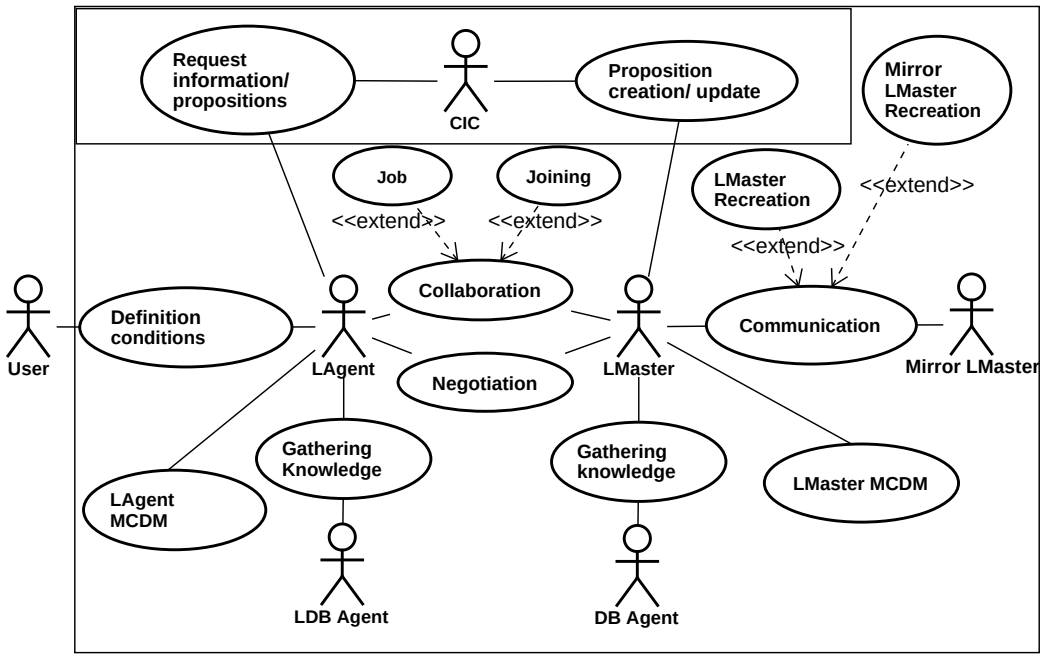

Figure 1. Use Case diagram of the proposed system

obtain a list of agent teams that satisfy its predefined criteria (currently, an exact match is required). Upon receiving such a list, due to trust considerations (see [3] for description of a trust management scenario that is directly applicable in the proposed system) it may remove certain teams from the list. For instance, a team that did not pay its workers will not be contacted again. The *LAgent* will then communicate with *LMasters* of the remaining teams. It will utilize the Contract Net Protocol and Multi Criterial Analysis to evaluate obtained proposals. If the *LAgent* finds a team that it would like to work with, it will join it. If no such team is found, the *LAgent* may decide to abandon the task and inform about it its *User*. It is also possible that the *LAgent* may decide to become the *LMaster* of a new team and follow steps involved in such process.

Let us now devote our attention to answering the question what happens when the *"User"* requests that its *LAgent* arranges execution of a task.

3. Selecting the team to execute the job

The first step to execute a job is for the *User* to provide its *LAgent* with the necessary information such as a job description, negotiation parameters and, possibly, constraints. Based on *User*-input, the *LAgent* acts autonomously trying to execute the job utilizing resources available within the Grid. As specified above, the overall schema of interactions is the same as in the case of *User* requesting its *LAgent* to join the team. First, the *LAgent* queries the *CIC* to obtain list of *agent teams* that have required resources for the job (again, only exact matches are returned). Then it utilizes trust information to filter-out untrustworthy teams (e.g. a team that last time did not satisfy the service level agreement, and did not deliver results on time, will, most likely, be removed from the list). Based on the filtered list of agent teams, the *LAgent* starts negotiations with their leaders (*LMasters*). During negotiations the *LAgent* utilizes the negotiation parameters and constraints specified by the *User*. The team that will execute the job is selected on the basis of Multi Criterial Analysis (MCA) [7]. Let us now look into details of the selection process.

3.1. User Input

User provides its *LAgent* with the job description, negotiation parameters and, possibly, execution constraints. In our approach we assume that the system will utilize ontologically demarcated data. Before we proceed, let us make the following remark. An ideal situation, for the development of our system, would be if there existed an all-agreed "ontology of the grid" (that

would include both the resources and job execution parameters). Unfortunately, while there exists a number of (separate and incompatible) attempts at designing such an ontology, at this stage they can be treated only as a "work in progress." 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 then 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. Obviously, when the grid ontology will be agreed on, our system *will be ready* for it. In [9] we have presented our ontological representation of computational resources. Here, let us focus our attention on the negotiation parameters which are expressed using, what we named, *Grid Yellow Pages Ontology*. Currently, in our work we utilize three negotiation parameters: cost, job start time and job end time. For each of theses parameters the user specifies its importance by assigning weight that is later used in the MCA (section 3.3). In addition to negotiation parameters we specify execution constraints, e.g. the maximum price that can be charged for the job. Furthermore, we assume that if any of job execution parameters should not be taken into account, then either 0 weight is given or that parameter is not included in the negotiation parameter-set. Alternatively it is also possible that a given parameter is constrained but it is not given a weight. This also shows how ontology-based approach gives us flexible possibilities of expressiveness. The following OWL Lite code snippet represents how these concepts are combined into the negotiation parameter-set.

```
### negotiation parameters ###

: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.

### generic constraints ###

:NegotiationParamConstraint a owl:Class.

:FloatConstraint a owl:Class;
  rdfs:subClassOf:NegotiationParamConstraint.

:maxFloatValue
  a owl:FunctionalProperty , owl:DatatypeProperty;
  rdfs:domain:FloatConstraint;
  rdfs:range xsd:float.

:minFloatValue
  a owl:DatatypeProperty , owl:FunctionalProperty ;
  rdfs:domain:FloatConstraint;
  rdfs:range xsd:float.

:TimeConstraint a owl:Class;
  rdfs:subClassOf:NegotiationParamConstraint.

:minDateValue
  a owl:DatatypeProperty , owl:FunctionalProperty ;
  rdfs:domain:TimeConstraint;
  rdfs:range xsd:dateTime.

:maxDateValue
  a owl:FunctionalProperty , owl:DatatypeProperty;
  rdfs:domain:TimeConstraint;
  rdfs:range xsd:dateTime.
```

As shown in the ontology schema, we separated concepts of constraints and parameters—they are defined in separate classes. This allows us to reuse constraints concepts definitions throughout different parameters definitions. For example, DateConstraint is used by JobStartTime and JobEndTime parameters. Currently, via constraints, we can define maximum or minimum value for our parameters; e.g. maximum cost or minimum jobStartTime. Note also that extending this parameter-set by adding, for instance, penalty for not completing job on time, requires only a relatively simple operation of extending our ontology and making minimal changes in agent-codes. This being the case, the focus of our work was on the agent interaction and parameter / constraint utilization, rather than development of a truly realistic parameter-set. Let us now as-

sume that the *User* stated that the cost of the execution is twice as important than the job end time by giving weight 2 to the cost and weight 1 to the job end time. Then the instance of the proposed parameter-set would have the form:

```
@prefix nego:
  <http://gridagents.sourceforge.net/Negotiation#> .

:NegotiationSetInstance a nego:NegotiationSet ;
  nego:negotiationParam [
    a nego:JobEndTime ;
    nego:paramWeight "1.0"^^xsd:float
  ] , [
    a nego:Cost ;
    nego:paramWeight "2.0"^^xsd:float
  ] .
```

In addition to the job-execution related parameter-set, the user specifies the resource describing parameters that are used to query the *CIC* (for more details, see [9]). As a way for the *User* to communicate input parameters to its *LAgent*, we have implemented a User Agent GUI (see section 4).

Let us now assume that, in response to the query, the *LAgent* obtained from the *CIC* the list of agent teams that have resources necessary to complete the job and has filtered these that are not worthy of its trust, and proceed to describe the *LAgent-LMaster* negotiations.

3.2. Negotiations

The *LAgent* utilizes the FIPA Contract Net Protocol to negotiate with *LMaster*s (see [1] for details). In Figure 2 we depict the a slightly adjusted version of the Contract Net Protocol, as it is pertinent to our situation. Note that the *LAgent* negotiates with more then one *LMaster* and therefore the same set of interactions takes place in all of these negotiations.

In the initial step, the *LAgent* sends out the *CALL-FOR-PROPOSAL* (*CFP*) message to all *LMaster*s on the final list (after pruning). The *CFP* contains the job description and the execution constraining parameter-set, according to which *LMaster*s are able to construct their offers (obviously, weights assigned by the *User* to individual parameters are not communicated). In the case of our simplistic parameter-set, the *CFP* message could have the following form:

```
(CFP :sender
  ( agent-identifier
    :name ua@kameleon:1099/JADE
    :addresses (sequence http://kameleon:7778/acc )
    :X-JADE-agent-classname UserAgent )
  :receiver (set ( agent-identifier
     :name lmaster@e-plant:1099/JADE
     :addresses (sequence http://e-plant:7778/acc )))
  :content  "((action
     (agent-identifier :name lmaster@e-plant:1099/JADE
    :addresses (sequence http://e-plant:7778/acc))
    (JobRequest :resRequirements
    (OntoData
      :ontoDataLang RDF/XML-ABBREV
      :ontoDataStr
```

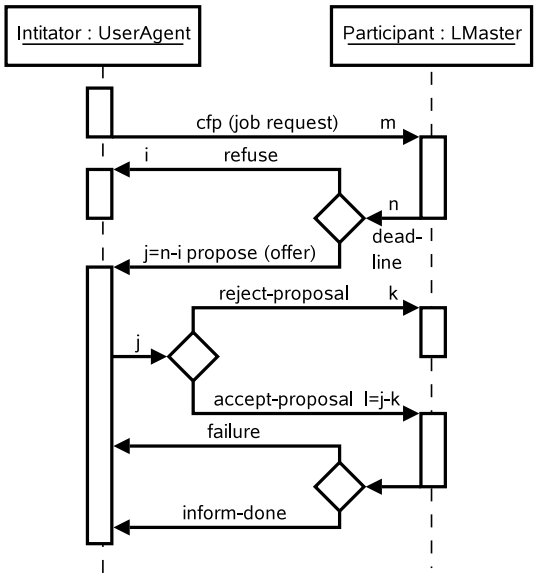

Figure 2. Interaction Diagram of FIPA Contract Net Protocol.

```
\"<rdf:RDF xmlns:grid="..." xmlns:rdf="...">
  <grid:UnitaryComputer
    rdf:about="jade://request@kameleon:1099/JADE">
  <rdf:type rdf:resource=
  "http://.../Grid#ComputerSystem"/>
  <grid:cpu>
    <grid:cpuClockSpeedMhz
       rdf:datatype="http://.../XMLSchema#int">
        1500
    </grid:cpuClockSpeedMhz>
    </grid:cpu>
  </grid:UnitaryComputer>
</rdf:RDF>\"
       )
     )
    :negoParamSet
      (NegotiationSet :negotiationParam
        (sequence (JobEndTime
          :jobEndTimeConstraint (TimeConstraint
          :maxDateValue \"2006-10-12T12:00:00\")))
          ))))"
 :reply-with  R1166131532630_0
 :language  fipa-sl0
 :ontology  Messaging
 :protocol  fipa-contract-net
 :conversation-id  C4916061_1166131532629 )
```

In this example the *LAgent* is looking for single machine with a 1.5MHz CPU and specifies the deadline for the job execution by constraining the jobEndTime negotiation parameter.

On the basis of the received *CFP* and their view of the situation on their teams, each *LMaster* prepares its proposal and sends it back to the *LAgent*, using a *PROPOSE* message. Note that it is possible that some of *LMasters* refuse the *CFP* (using a *REFUSE* message). For example, in the time between the last update of team information in the *CIC* and the *LAgent*'s request, some resources "disappeared" and the team cannot complete the task. The positive response message, containing an

offer could have the following form:

```
(PROPOSE
  :sender ( agent-identifier
    :name lmaster@e-plant:1099/JADE
    :addresses(sequence http://e-plant:7778/acc)
    :X-JADE-agent-classname LMaster )
  :receiver (set (agent-identifier
    :name ua@kameleon:1099/JADE
    :addresses(sequence http://kameleon:7778/acc)
    :X-JADE-agent-classname UserAgent ) )
 :content "(
    (result
      (action
        (agent-identifier
          :name lmaster@e-plant:1099/JADE
          :addresses(sequence http://e-plant:7778/acc)
          )
    (JobRequest
           ...
          )
      )
      (JobRequestOffer
        :negoParamSet
          (NegotiationSet :negotiationParam
          (sequence
          (JobStartTime
             :jobStartTimeValue\"2006-10-12T11:30:00\")
          (JobEndTime
              :jobEndTimeValue\"2006-10-12T12:30:00\")
            (Cost
              :costValue \"120")
            )
          )
        )
    )
  )"
  :reply-with ua@kameleon:1099/JADE1166137976099
  :in-reply-to R1166137976093_0
  :language fipa-sl0
  :ontology Messaging
  :protocol fipa-contract-net
  :conversation-id C4916061_1166137976092
)
```

The response message informs the *LAgent* that the *LMaster* is willing to complete the job and devote resources to it within the specified time-frame and that the total cost will be 120 units. Note that in the Contract Net Protocol, sending a *PROPOSE* message constitutes a *commitment* of the *LMaster* to the conditions it specified.

In the current design of the system the *LAgent* awaits for responses until all of them arrive or a specific deadline occurs. Note that it is necessary to impose a deadline to avoid a deadlock; e.g. it is possible that one of *LMaster*s looses connection to the Internet and cannot communicate back its offer. If after the deadline there is no proposal then the *LAgent* cannot execute the task and reports this fact back to the *User*. Otherwise, if there is at least one offer, the *LAgent* starts evaluating offers. Proposal evaluation is a two-stage process:

- Offers which do not meet execution constraints (e.g. cost, job start time, job end time) are filtered out. If all offers are filtered out at this stage, due to constraints, then the *LAgent* cannot execute the task and reports back to the *User*.
- The remaining offers are evaluated using Multi Criterial Analysis (*MCA*)—see section 3.3.

The first stage of the process requires an explanation. It is reasonable to ask: why would an *LMaster* send an offer that violates constraints that were given to it. This situation is, on the one hand, result of a simplification in our current design of the system; while on the other hand it is a preparation for future system extensions. The simplification concerns the *LAgent*, which filters out all offers that violate constraints. Observe that this may result in very few, or no offers at all. This also prevents the *User* from specifying *soft constraints* that represent a "strong preference" but violation of which does not necessarily mean that the offer is unacceptable. For instance, I may prefer to have this job done tonight, but if I can have it done extremely cheap by tomorrow evening, then I may be willing to accept this offer. On the other hand, the preparation for the future system extension is on the side of the *LMaster*. Its behavior is being prepared for job constraints that may be "flexible" (here, it is important to note, that some constraints may actually be "sharp" and their violation may result in an offer being filtered out; currently we have not decided if the *LMaster* is going to be informed if a given constraint is flexible or not, but we are being swayed toward the solution in which the *LAgent* keeps this information to itself). Now, recall that the *LMaster* has knowledge of the capability of its team and its "pricing policy" and when it makes an offer, it is going to be one that can be backed up by a service level agreement. Let us assume that an *LMaster* may have a full load for the next 12 hours, but then has no jobs scheduled. In this case it may make an offer which is going to violate the timing constraint—as the execution will start past the suggested deadline—-but since it has no tasks scheduled, it may make an extremely cheap offer. We plan to address these types of reasoning and strategizing in the future.

After the *MCA* is applied to the remaining offers, the specific team is selected to execute the job. In this case the *ACCEPT-PROPOSAL* message is sent to the *LMaster* of that team. The remaining teams are rejected by sending to them the *REJECT-PROPOSAL* message. The selected team confirms acceptance by sending back an *INFORM-DONE* message. Obviously, the Contract Net Protocol is also taking care of various "emergency situations;" e.g. failure of the selected team to respond.

3.3. Multi Criteria Analysis

Let us now describe in more detail the Multi Criterial Analysis-based selection process. In the current implementation of the system we use the *linear additive model* [7]. Note that this model was selected for its

simplicity. However, it has to be stressed that any other *MCA* method can be applied to evaluate received offers.

In the case of the linear additive model, evaluation is done by multiplying value scores on each criterion by the weight of that criterion, and then adding all those weighted scores together. Recall, that we have three criteria that take part in the MCA process: cost, job start time and job end time. If communication with m teams resulted in n proposals ($m - n$ teams refused, send us proposals that were filtered out, or did not respond within the deadline) then criterion scores of the i-th team are calculated as follows:
Start Time Score:

$$STS_i = \frac{(startTime_i - currentTime)^{-1}}{\sum_{j=1}^{n}(startTime_j - currentTime)^{-1}}$$

End Time Score:

$$ETS_i = \frac{(endTime_i - currentTime)^{-1}}{\sum_{j=1}^{n}(endTime_j - currentTime)^{-1}}$$

Cost Score:

$$CS_i = \left(\cos t_i \sum_{j=1}^{n} \cos^{-1} t_j\right)^{-1}$$

All scores are normalized and the i-th team final score is calculated as:

$$\text{Team Score } TS_i = STS_i startTimeWeight + ETS_i endTimeWeight + CS_i costWeight$$

Team with the highest overall score, obtained as a weighted sum of individual criterion scores, is selected as the "winner." For the example of the MCA in use, please refer to the next section.

4. Example

Let us now present an example of how our system works to support a *User* who would like to execute a job utilizing the MPI programming library on 16 processing nodes of a single computer. First, our *User* would specify resource requirements using the GUI interface shown in the Figure 3.

In the next step, the *User* has to provide its *LAgent* with negotiation parameters expressing her execution preferences. For example, let us assume that she would like to meet the deadline of 12:00 hours on 11th of October, 2006. Furthermore the cost should be no larger that 100 units. Note that the cost does not matter as much as the time—the time weight is 3, while the cost weight is 1; see Figure 4). Finally, in this example

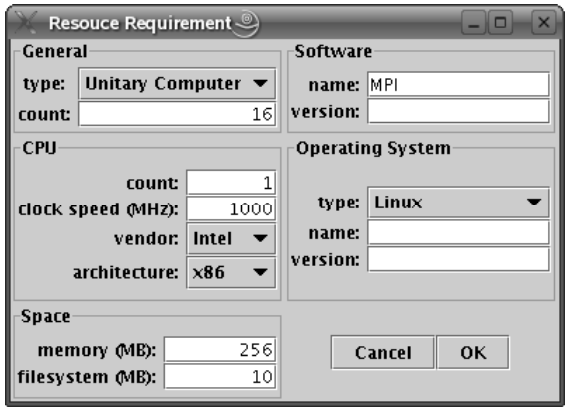

Figure 3. User Agent GUI: Resource requirements.

Figure 4. User Agent GUI: Weights and constraints of criterions.

the *User* does not care when the job starts (weight 0), she only wants to meet a specific execution deadline.

From the specified resource requirements the *LAgent* creates the ontological instance of resource requirements and, based on that instance, constructs a SPARQL query (utilizing *Resource Requirements to SPARQL* translator module *ResReqToSPARQL*) and sends it (as an ACL request message) to the *CIC* agent. The *CIC* executes the query and as a result delivers a list of candidate teams meeting resource requirements. The answer from the *CIC* contains a list of contacts to *LMasters* representing teams, encapsulated in a *ResultSet*.

Since trust management is not yet implemented, the *LAgent* starts Contract Net Protocol-based negotiations with all *LMasters* found on the list. It receives their offers, filters these that violate constraints and evaluates the remaining ones using the MCA. Figure 5 shows three teams and their scores evaluated by the MCA.

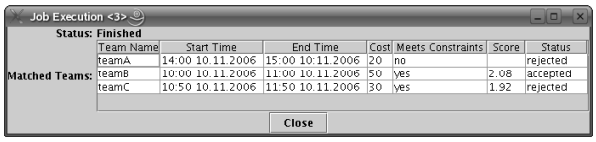

Figure 5. User Agent GUI: Matched teams and their scores.

As we can see, *teamA* has been rejected because it does not meet the deadline constraint. The remaining two teams have been evaluated. Despite of cost proposed by the *teamC* being cheaper by 40%, it was the *teamB* that was accepted to do a job because of an earlier job completion time. Note that, unless instructed otherise, the decision is made by the *LAgent* autonomously and the depiction in Figure 5 is presented only to illustrate the process.

5. Concluding remarks

In this paper we have continued describing our work devoted to development of an agent-based grid resource-brokering system. Here we have focused our attention on the process involved in an agent representing a *User* selecting a team that is going to execute its job. We have presented how job-execution parameters and constraints are represented and how they are utilized in Contract Net Protocol based negotiations between *LAgent*s representing the *User* and *LMaster*s representing agent teams. We have also indicated how a much more involved forms of reasoning can be easily introduced into our design. Currently we are using experiences gathered from our initial implementation to re-implement parts of the proposed system to improve its efficiency and flexibility. We are also focusing our attention on the process of team-formation. We will report on our progress in the near future.

References

[1] Fipa contract net protocol specification.

[2] 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.

[3] 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.

[4] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, pages 1507–1542, 2002.

[5] G. Cabri, F. D. Mola, and R. Quitadamo. Supporting a territorial emergency scenario with services and agents: a case study comparison. In *The IEEE 15th WETICE*, Manchester, UK, 2006.

[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, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki. Efficient matchmaking in an agent-based grid resource brokering system. In *Proceedings of the International Multiconference on Computer Science and Information Technology*, pages 327–335. PTI Press, 2006.

[9] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki. Utilizing agent teams in grid resource management - preliminary considerations. In *Proceedings of the J. V. Atanasov COnference*, October 2006.

[10] 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, Washington, DC, USA, 2004. IEEE Computer Society.

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

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

[13] 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 3470/2005 of *Lecture Notes in Computer Science*, pages 651–660, Germany, 2005. Springer Verlag.

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

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

[16] 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.