Trust Management in an Agent-based Grid Resource Brokering System—Preliminary Considerations

Maria Ganzha[*], Marcin Paprzycki[†] and Ivan Lirkov[**]

[*] *Systems Research Institute*
Polish Academy of Science
maria.ganzha@ibspan.waw.pl
[†] *Computer Science Institute*
Warsaw School of Social Psychology
marcin.paprzycki@swps.edu.pl
[**] *Institute for Parallel Processing*
Bulgarian Academy of Science
ivan@parallel.bas.bg

Abstract.
It has been suggested that utilization of autonomous software agents in computational Grids may deliver the needed functionality to speed-up Grid adoption. I our recent work we have outlined an approach in which agent teams facilitate Grid resource brokering and management. One of the interesting questions is how to manage trust in such a system. The aim of this paper is to outline our proposed solution.

Keywords: Grid computing, agent systems, trust management
PACS: 02.70.-c

1. INTRODUCTION

Grid computing became one of promising approaches to utilization of heterogeneous, geographically distributed computers. Virtualization of resources facilitated by the Grid is expected to have a broad impact in science and business. Unfortunately, adoption of the Grid, while speeding-up recently, is still unsatisfactory. One possible reason for this situation is rather complicated support for resource brokering and management provided by the current Grid infrastructure. At the same time, it has been suggested that autonomous software agents combined with semantic description of resources may be a step in the right direction [1, 2]. Finding these arguments compelling, we have searched for existing solutions that match this vision. Results of our search have been summarized in [3] and showed that existing solutions are somewhat limited in scope and robustness. Therefore, we have proposed a different approach in which agent teams collaborate to fulfill user requirements.

Specifically, [4] contained an initial overview of the proposed approach. In [5] we followed with a study of effective ways of implementing yellow-page based matchmaking services. In [6] we have considered processes involved in agents seeking teams to execute their jobs, while in [7] we have discussed how agents join teams (agent teams are formed). The aim of this paper is to conceptualize processes involved in trust management that take place in the proposed system. Before we proceed let us make explicit some assumptions that underline our work.

1. First, we follow these who state that software agents will play an important role in design, implementation and long-term upkeep of large-scale software systems (see e.g. [8]). While we do acknowledge that this claim is not uncontroversial and that there are these who see software agents as just repackaging of ideas existing in software engineering and distributed system for years [9], we do not see our role as becoming involved in such fundamental discussion. Our role is to utilize existing tools and environments (such as JADE agents [10]) to develop, implement and experiment with agent systems.
2. Second, we assume that software agents will be directly involved in the future development of the Grid. In this way we follow (and accept) arguments put forward in the aforementioned [1, 2]. Note that these arguments are further supported by the body of research devoted to combining software agents and Grids (summarized in [3]). Therefore, we are immune to challenges based on the claim that agents may not be necessary / needed.
3. Next, note that there are two ways in which the Grid can be approached. The first views it as a *local infrastructure* and includes Grids within a university, company and even large scale Grids funded by the EU. The main distinguishing factor is that here there exists a more-or-less centralized control structure. Note that even in an extremely large grid structure like the EGEE, each node is controlled by local administrators and the overall structure is also monitored to assure that some form of quality of service (QoS) control is enforced. In other words, existence of service level control mechanisms and of a centralized "authority" distinguishes what we dubbed the *local Grid*.
   The second approach to Grid is by viewing it as a global infrastructure consisting of nodes contributed by individual users; view stemming from ideas outlined in [11]. In this case no centralized (or even localized) control of quality of service is available. For instance, any power interruption to a home PC causes it to "go down" taking with it jobs that have been executed. Similarly, PC user starting to play Doom, or watching movies on her PC overloads the processor and thus reduces the total amount of resources available to Grid users. While such problems have not been a problem in early large scale distributed applications, such as SETI@HOME or United Devices Search for the Cure for Cancer, they become an immediate problem when results have to be produced in a certain order and deadlines are crucial.
   In our work we are interested in the second scenario, where the Grid is viewed as a collection of nodes contributed by individuals, and possibly organizations, and thus issues of quality of service and enforcement of service level agreement play a very important role.
4. Following work of R. Buyya [12] we view the computational Grid as an environment in which workers (in our case *agent workers*) contribute their resources (programs, data, computational power, etc.), and are remunerated for their usage, whereas users (in our case *agent users*) utilize available resources to complete their tasks, and want to pay a fair price. Furthermore, they want to decide about issues like: should they pay less for longer time or pay more for an express job.

5. We assume that ontological demarcation (particularly of Grid resources) and semantically-oriented information processing are in the future of the Grid. Note that we do not involve ourselves in the discussion of long term feasibility of ontologies as a solution to the problem of integration of heterogeneous resources. Rather, we utilize them in our work.
   Obviously, currently the WSDL language ([13]) and Grid service description standards defined by the OGSI [14] are the most popular ways of demarcating Grid services. However, the fact that they are based only on XML makes them much less useful (looking into the future) than full blown RDF/OWL based ontologies (especially when semantic reasoning / matchmaking is utilized). We also recognize efforts like: (1) Earth System Grid (ESG) [15], which is a mixture of Grid-oriented and Earth Science-oriented ontologies, (2) Core Grid Ontology (CGO) [16], which tries to deal with the most basic grid concepts, (3) Web Services Resource Framework (WSRF) [17], which defines key concepts within the Globus framework and is geared toward integration of the Grid and Web Services, (4) Agent Grid Integration Ontology (AGIO) [18] which ontologically defines concepts involved in agents being introduced into Grids, or (5) ontologies defined within project like Unicore and GLUE. However, at this stage all these efforts can be treated only as "work in progress." Therefore, instead of selecting one of them, we focus our work on the agent-related aspects of the system (designing and implementing agent system skeleton) while utilizing very simplistic ontologies). Obviously, when the Grid ontology will be agreed on, our system *will be ready* for it; as only parts that deal directly with ontologies and ontological reasoning will have to be adapted.

Keeping these assumptions in mind we proceed as follows. In the next section we start with an overview of the proposed system based on its UML Use Case Diagram. We follow with the general considerations involved in trust management and use them to discuss in more details the four specific situations when agents evaluate trustworthiness of each other.

2. PROPOSED SYSTEM

The above described set of assumptions became the starting point of designing our system. To be able to (a) facilitate work in a global Grid, (b) assure some form of service level agreement and thus control of quality of service, and (c) support Grid economy; we propose virtual organizations, called *agent teams* based on the following rules:

- agents work in teams (groups of agents)
- each team has a leader / manager—*LMaster agent*
- each *LMaster* has a mirror *LMirror agent* that can take over its job in case when the *LMaster* "fails"
- incoming workers (*worker agents*) join teams based on their own individual set of criteria
- teams (represented by their *LMaster*s) accept workers based on an individual set of criteria (specific to each team)

- each *worker agent* can (if needed) play the role of an *LMirror* (and thus of an *LMaster*)
- matchmaking is ontologically grounded and utilizes yellow page type approach [19] (facilitated by the *CIC* infrastructure; see also [20])

The proposed approach can be thus represented in the form of a Use Case diagram depicted in Figure 1. Let us now briefly describe agent roles and interactions taking place in

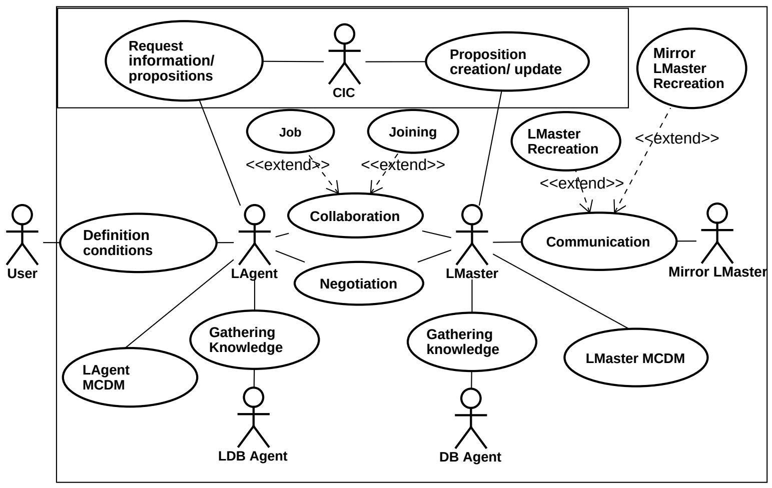


FIGURE 1. Use Case diagram of the proposed system

two scenarios: (1) agent looking for a team to execute a task and (2) agent looking for a team to join (more details can be found in [4, 5, 6, 7]). We start from the *Client Information Center* (*CIC*) that provides matchmaking services. As described in [19], there exist multiple ways to facilitate matchmaking in a distributed system, and each one of them has specific advantages and disadvantages. In our work (and following our experiences described in [20]) we have decided to utilize a *yellow page*-based approach and thus the *CIC* infrastructure stores team advertisements containing information about offered resources (e.g. hardware capabilities, available software, etc.) as well as information that a given team is searching for members with specific (hardware, software etc.) characteristics. Note that this information is stored in an ontologically demarcated fashion utilizing our simple Grid resource ontology and is persisted in a Jena [21] repository.

To describe dynamic processes depicted in static form in Figure 1, let us assume that the system has been running long enough for some agent teams to be formed and to advertise their services / needs in the *CIC*. Note that the *User* can either want to contribute its resources as a team member, or utilize the Grid to complete a specific task (in the Use Case diagram both situations are “UML-symmetric”).

User who wants to contribute resources (and be paid for doing so) formulates conditions for joining a team and communicates them to its agent (the *LAgent*). The *LAgent* requests from the *CIC* list of teams that look for members and satisfy the predefined criteria. Upon receiving such a list, due to trust considerations (see Section 3.2.3) it may remove certain teams from the list. Next, the *LAgent* communicates with *LMasters* of the remaining teams and utilizes FIPA Contract Net Protocol [22] and multicriterial analysis

[23] to evaluate obtained proposals. The result of such interactions may be twofold: (1) the *LAgent* finds a team to join (and joins it), or (2) no such team is found (either there was no acceptable offer or no offer at all). In the latter situation the *LAgent* informs its *User* and awaits further instructions.

When *User* requests that its *LAgent* arranges execution of a task, the scenario is very similar. First, *User* communicates to its *LAgent* conditions of task execution. Then the *LAgent* queries the *CIC* to find out which teams can execute its task. Upon receiving a list of teams the *LAgent* removes from it teams that cannot be trusted (section 3.2.2). Next, it communicates with *LMaster*s of the remaining teams and utilizes FIPA Contract Net Protocol and multicriterial analysis to find the best team to execute its job. Note that if no team will satisfy specified conditions, the *LAgent* reports this situation to its *User* and await further instructions.

3. TRUST MANAGEMENT

3.1. Preliminary considerations

Let us start from precisely identifying our interests. First, the notion *trust* is very often associated with *security*. However, let us assume that a Grid node is secure (and can be safely used). Does this mean that it will be considered trustworthy? Not if such a node will go down at random intervals multiple times a day. In this case it is not likely to be used and will not be considered trustworthy. This clearly indicates that $trust \neq security$. In what follows we assume that *security* is assured, and focus on other aspects of *trust*.

Second, let us distinguish two notions: *trust* and *reputation*. *Trust*—a peer's belief in another peer's capabilities, honesty and reliability based on *its own* direct experiences. *Reputation*—a peer's belief in another peer's capabilities, honesty and reliability based on recommendations received from *other peers*. Thus, *trust* is typically conceptualized as a one-to-one relationship, while *reputation* is a one-to-many relationship. Furthermore, both *trust* and *reputation* are based on long-term relationships and are cumulative.

Let us now observe that the need for trust management typically does not arise within *local Grids*. A Grid within an organization, or a collection of organizations that are bound by an agreement makes trust considerations unnecessary. However, in the latter case, when a particular node is down more often than expected by partners, it may be removed from the consortium. It is also very likely that local Grids do not involve “real” financial transactions (it is “virtual money” that moves between corporate departments). Therefore, we focus our attention on *global Grids* and start from their economical aspect.

In this context, recent report produced by Insight Research claims that the economic value of the Grid will reach $25 billion by the year 2011 ([24]). Such move of the Grid into mainstream business can be illustrated by Schlumberger selling Oil-related Grid services through its Grid Portal [25], or Sun Microsystems selling computer time (for $1 an hour) through the Grid infrastructure [26].

Obviously, to facilitate paying per usage it is necessary to utilize accounting software. The Open Grid Service Architecture (OGSA) [27] has introduced four services that can be used to develop support for economical transactions: (1) *metering service*, which measures resource usage, (2) *rating service* that translates usage into chargeable fees, (3) *accounting service*, which associates payment with a specific user, and (4) *billing service* that interacts with the “outside world” to finalize payments. There exist also other Grid accounting services, such as: (a) Grid Service Accounting Extensions [28], (b) Grid Economic Services Architecture [29], (c) SNUPI [30], or (d) GridBank [31]. In each case the Grid node runs a monitoring service that reports resource usage used for billing purposes. Interestingly, as shown in [32], all these methods are based on an unrealistic assumption that parties involved in Grid computing will not cheat to gain monetary advantage (e.g. by manipulating the operating system kernel to mislead the accounting software). In response authors of [32] proposed a method based on a trusted authority that will empirically “verify” fidelity of Grid nodes (by running random jobs on randomly selected machines). Unfortunately, (i) someone would have to pay for existence of such entity (and benchmarking jobs it would execute), (ii) its reliability remains an open question taking into account existing variety of computer hardware and software combinations, and (iii) scalability of the proposed solution becomes a real issue when the number of node increases (will the approach work for more than 10K nodes in the Grid?).

In this context we would like to propose that instead of measuring actual usage (which, as shown in [32], may be intentionally manipulated), it is possible to utilize a “resource rental” model. In this case, like in a case of car rental, payment is associated with length of rental, regardless of the actual usage (the same way as week-long car rental with unlimited mileage costs $156.78 regardless if the car was driven 18 or 1800 miles). Note that this is also the basic model of team member availability proposed in [7].

3.2. Proposed approach

The above considerations allow us to introduce trust management into the system described in Section 2. Note that the proposed solution is geared to work within that specific system and this directly influences our approach. As discussed above (in Section 2), there exist four situations that involve trust-related considerations.

1. When the *LAgent* obtains the list of teams that it can join, it checks if they are trustworthy; for instance if it worked with a given team and the terms of agreement were not fulfilled (e.g. it was promised that it will be utilized at least 30% of time, but it was not the case), then the *LAgent* may not want to work for a given team.
2. When the *LAgent* obtains the list of teams that can execute its task, it checks if they are trustworthy; for instance if a given team promised to complete the task within 5 hours and did not, then the *LAgent* may not want to work with a given team.
3. When an *LMaster* receives a call for proposals from an *LAagent* that wants to join its team, but this *LAgent* broke an agreement in the past (e.g. it was not available everyday between 10 PM and 6 AM), then it may not want to work with it.
4. When an *LMaster* receives a call for proposals from an *LAagent* that tried to avoid paying for the last job, then it may not want to get to business with it.

Let us now discuss these four scenarios to establish how trust materializes in each case and how it can be managed in our system.

3.2.1. *Team evaluating incoming user*

This scenario is the easiest to conceptualize. User is evaluated on the basis of its past behaviors. When user is "renting" the resource, the situation is extremely simple. The only way the user could cheat is if he did not pay for service. Obviously, such situation has been recognized for some time in e-commerce and resulted to creation of proxy services (e.g. PayPal) that assure that payment is released after service is delivered. Therefore, user can be "forced" to pay for the service (as long as the service provider can show that the service was actually made available).

The situation becomes somewhat more complicated when the pay-per-use model is applied (see [32] for more details). In this case user can claim that she was cheated by being billed for more than actual execution time. However, in such a case the resource provider can use full disclosure of its hardware, software and system logs to respond to accusations (including job re-run). Obviously, provider that was falsely accused of cheating may decide in the future to not to work with a given user.

3.2.2. *User evaluating service providing team*

In the opposite case, when the *LAgent* evaluates a team to establish that it is trustworthy, it checks past fulfillment of the service level agreement. In the first model (service rental) the SLA can be very simple and state that a specific resource is to be available from 23:25 till 6:30. In this case the actual availability of the resource fulfills the contract. In the case of a multi-feature SLA lack of satisfaction of any feature may result in loss of trust. Furthermore, penalty for each violation may differ in severity. For instance, delivery of a machine with 1 Gbyte of RAM instead of 2 Gbytes of RAM may result in a lesser penalty than delivery of a machine with 2.7 GHz processor instead of a 3.2 GHz processor; depending on the type of job to be executed and the (claimed by the user) effect that these these changes have on performance. Note that the actual performance loss may be almost invisible, while the user may penalize the team according to its own beliefs (and the penalty value remains private to the user). Based on results presented in [33] we can propose the following approach to measuring trust level. Let us assume that contract fulfillment will be denoted e_0, while each breach of contract (where we distinguish m such events) will be denoted e_i, for $i = 1, m$. Now, each user will assign weight $TA(e_i) \in [-1, 1]$ (where $i = 0, m$) to each event e_i. Note that each user may assign different value to each event (and even distinguish different set of events). Let us now assume that after n interactions between a given user and a given team (X), the trust value is $T_n(X)$. Then, assuming that the result of interactions was event e_k (where $0 \le k \le m$) the trust value after the $(n+1)$-st interaction will be calculated as follows

$$T_{n+1}(X) = (1 - \alpha |TA(e_k)|) T_n(X) + TA(e_k), \qquad (1)$$

where $\alpha \in (0, 1)$ is the sensitivity parameter. Specifically, when α is close to one then each event (positive or negative) has an direct effect on the trust score. Therefore just a few missteps makes a high trust score to fall down sharply. Similarly, few fulfilled contracts make it raise fast. This represents a "frantic" user. On the other hand, for α

close to zero both positive and negative events have only a limited effect on the total score. This type of a “phlegmatic” user has a stable trust image of others; one that is difficult to improve or tarnish.

The trust value is used by the *LAgent* in two situations: (1) to prune teams that are untrustworthy, (2) in multicriterial analysis to establish which team to interact with. In the first case each user establishes a specific threshold value below which it deems a given team untrustworthy and removes from considerations. IN the latter case trust value is one of weighted criteria used in the selection process. Let us note first that when user is confronted with a team it has never interacted with, (unless reputation is introduced into the system) it will assign it a default trust value, which is slightly above the threshold. Second, following [34], we propose trust model with forgetting. Therefore, each extended lack of interaction pushes the trust value toward the default one. Finally, let us note that the proposed trust management schema does not depend on the particular form of usage billing (rental or actual usage-based). In both cases user can define specific breaches of the SLA and penalties associated with each of them.

3.2.3. Worker evaluating team

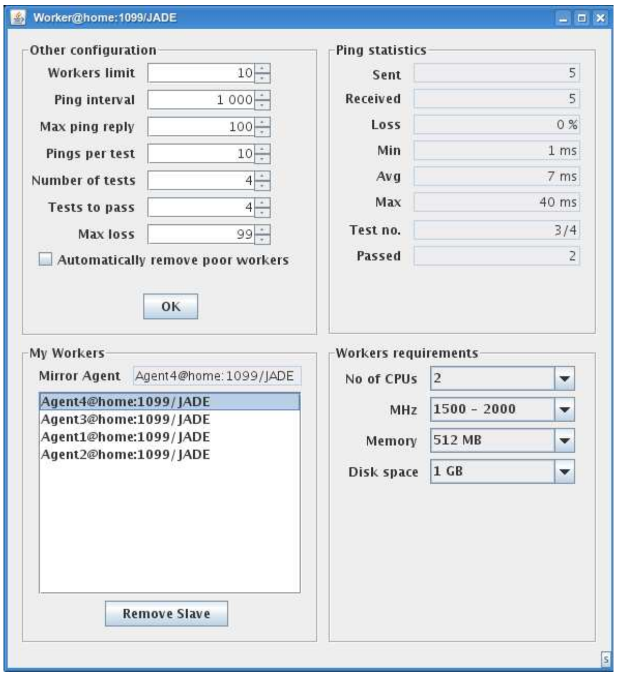
Here, evaluation of trust depends on the economic model used. The simplest situation is when worker sells “time of availability” and is paid for being available, while the actual work done does not directly affect the payment. Then the only thing that the *LMaster* has to check is if the node is “alive.” This approach has advantages and disadvantages. Its main advantage is that it is immune to cheating mechanisms described in [32]. Furthermore, it protects the team from effects of fluctuations of workload. On the one hand, the team has enough workers when the workload is high, on the other workers are shielded from effect of lack of jobs. In this case we can see a direct usage for the “overhead” collected by the *LMaster* (see [7]). For instance, while the market value of a given machine (based on its processor, memory and disk space) is 80 cents per hour, its *LAgent* will be contracted at 60 cents an hour, while the remaining money earned will be used to sustain the team while there is no work. Main disadvantages of this approach are: (1) *Lagent*s may join multiple teams and be paid for doing nothing; (2) workers are paid even if they do not work.

In the case when workers are paid for actual work done an important issue is how much work was actually there (the more work there is the better for the team members; unless there is too much work and they cannot handle it and the team will be penalized by users for breaching contracts). Here, the amount of work can be a part of the contract between the team and the worker. For instance the SLA can specify the the team guarantees that at least 30% of the contracted time will be spend working and earning money. Also more complicated clauses can be a part of the SLA. Note that the above introduced model of trust management can be directly applied here. Regardless of the specific form of contract, as soon as the SLA is finalized, worker can establish a system of rewards and penalties and apply formula 1 to calculate the level of trust. The remaining parts of the schema follow. Each user has a threshold value and a default trust value. After each experience of being a member of the team the trust value is updated, while after a long period without interactions the trust value shifts towards the default trust.

Note that the same team is likely to be evaluated twice: (1) as a team to work for, and (2) as a team to do the work. Interestingly, the two trust scores may be completely different. Team *X* evaluated as a team to work for may have a very high trust value, while the same team evaluated as a team that executes a task may have a low trust value. While it could be interesting to conceptualize the way that the two trust values could be used jointly, we leave this issue open and assume that the two values will be kept separate.

3.2.4. Team evaluating worker

The last case of interactions is *LMaster* evaluating its workers. The main assumption presented in [7] was that the contract between worker and the team involves stipulation that a given agent will be available at certain time periods. To check such availability we have proposed a specific mechanism depicted in Figure 2. Of interest for this paper is


Worker@home:1099/JADE

Other configuration

Workers limit	10
Ping interval	1 000
Max ping reply	100
Pings per test	10
Number of tests	4
Tests to pass	4
Max loss	99

☐ Automatically remove poor workers

Ping statistics

Sent	5
Received	5
Loss	0 %
Min	1 ms
Avg	7 ms
Max	40 ms
Test no.	3/4
Passed	2

My Workers

Mirror Agent: Agent4@home:1099/JADE

Agent4@home:1099/JADE
Agent3@home:1099/JADE
Agent1@home:1099/JADE
Agent2@home:1099/JADE

Workers requirements

No of CPUs	2
MHz	1500 - 2000
Memory	512 MB
Disk space	1 GB

FIGURE 2. LMaster pings non-stop his worker

the information presented in the *Other configuration* box (description of the remaining information can be found in [7]). To assesses the state of each team member the *LMaster* is continuously performing monitoring sessions. Each such session consists of a certain number of tests (parameter *Number of tests*; in our case 4), while each test consist of a certain number of pings (parameter *Pings per test*; in our case 10). Pings are send in an interval (parameter *Ping interval*; in our case 1000 milliseconds). A given ping is counted as a success if a response comes within predefined time (parameter *Max ping reply*; in our case 100 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 99% would result in failing the test; meaning that a test is failed only if all pings were not replied to). We have also specified the number of tests that a given *LAgent* has to pass (parameter *Tests to pass*; in our case, 4 tests in a round; in other words, all tests) to be considered to be a "live" worker agent. After completing a monitoring session, the *LMaster* zeroes all counters and a new such session starts.

While it is possible that a given team may require that a worker will be available all the time (must pass all tests by responding in time to all pings), such an approach seems somewhat unrealistic. It is obvious that for a variety of reasons a given worker may be cut-off from the network or go down temporarily. This can be included in the SLA and specify that a certain number of failed tests is allowed during the time of the contract. This allows us to distinguish three situations. During the time of contract the *LAgent* has violated it (vc), the *LAgent* has fulfilled the contract (fc), or it did more than just fulfill the contract (ae). The latter case means for instance that while the *LAgent* was allowed to be out of reach 7 times within a week, it was always available. Obviously, if the contract requires that the *LAgent* is always available only two situations are possible (vc and fc). However, observe that distinguishing the three situations allows us to collect information that can be very valuable when evaluating a potential worker for a job. Let us assume that for each worker we store a triple $(\#vc, \#fc, \#ae)$. In this case, instead of having only the trust value to utilize, we can establish how many times a given *LAgent* performed above expectations, fulfilled or failed to fulfill the contract. Now, nodes that are consistently characterized by the above expectations performance should be assigned to jobs that require extra assurance that they will be completed in time, while other nodes can be assigned low priority jobs.

Obviously, the proposed scoring method does not preclude utilization of the trust value, which becomes a composite representation of performance. Here, each result of *LAgent* being a worker for a given team can be assigned reward or penalty and utilized within formula 1. Furthermore, utilization of default trust, threshold trust value and trust-forgetting remain valid. What we suggest is that evaluating performance in three specific categories can supplement the composite trust value and help distinguish these *LAgent*s that are extra important to the functioning of the team from these that barely meet expectations. Note that in the composite value it is difficult to conceptualize "extra effort" as a high trust value may be a result of a large number of OK performances, or a result of a small number of brilliant performances.

3.2.5. Reputation

To complete this paper let us make a few remarks about the way that reputation can be used in the proposed system. First, let us note that all *LAgent*s have to be registered with the *CIC*. This means that it could be possible to create an eBay type ranking system. Specifically, each team (and possibly each agent) could have its score (based on opinions of team users, or teams evaluating workers) amalgamated within the *CIC*. In this case, when a potential worker approached a team, its *LMaster* could contact the *CIC* and obtain the combined reputation score and use it in preparing its response. Similarly, the *LAgent*, when asking the *CIC* for a list of teams to do the job or to join, could obtain not only the list, but also amalgamated reputation scores of each team, and use it to decide which teams are worthy sending the CFP to.

However, we could be interested not only in establishing a combined reputation score of each team / agent, but also in individual recommendations based on collaborative filtering. In this case we have to recognize that the Grid scenario discussed here involves *LAgent*s/*LMaster*s that may or may not be present to give their advice to the requesting *LAgent*. Therefore, an approach similar to the one discussed in [35] could be utilized.

4. CONCLUDING REMARKS

The aim of this paper was to conceptualize processes involved in trust management in the agent based resource brokering system that we are designing. First, we have identified two scenarios in which four cases of trust-based interactions take place. Second, we have described how trust materializes and can be quantified in each of them within two economic models: (a) resource rental, and (2) pay per use. Obviously, validity of all proposed models has to be established experimentally, which is what we plan to do in the near future.

ACKNOWLEDGMENTS

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

REFERENCES

1. 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*, IEEE CS Press, Los Alamitos, CA, 2004, pp. 8–15.
2. H. Tianfield, and R. Unland, *Multiagent and Grid Systems* **1**, 89–95 (2005).
3. M. Dominiak, M. Ganzha, M. Gawinecki, W. Kuranowski, M. Paprzycki, S. Margenov, and I. Lirkov, “Utilizing Agent Teams in Grid Resource Brokering,” in *Multiagent and Grid Systems*, 2007, in press.
4. 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*, IEEE CS Press, Los Alamitos, CA, 2006, pp. 46–51.
5. 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*, PTI Press, 2006, pp. 327–335.
6. 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*, IEEE CS Press, Los Alamitos, CA, 2007, pp. 249–256.
7. 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*, 2007, in press.

8. N. R. Jennings, “An agent-based approach for building complex software systems,” in *CACM*, ACM, 2001, vol. 44, pp. 35–41.
9. F. Kordon, Private communication (2006).
10. `http://www.tilab.jade.com`.
11. I. Foster, and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2003.
12. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, *Concurrency and Computation: Practice and Experience* pp. 1507–1542 (2002).
13. `http://www.w3.org/TR/wsdl`.
14. `http://www.gridforum.org/ogsi-wg`.
15. `http://www.earthsystemgrid.org/`.
16. W. Xing, M. D. Dikaiakos, and R. Sakellariou, “A Core Grid Ontology for the Semantic Grid,” in *6-th IEEE International Symposium on Cluster Computing and the Grid (CCGRID’06)*, 2006, pp. 178–184.
17. `http://www.globus.org/wsrf/`.
18. F. Douvert, C. Jonquet, P. Dugnie, and S. Cerri, “Agent Grid Integration Ontology,” in *Proceedings of the AWeSOMe’2006 Workshop*, Springer, LNCS, 2006, vol. 4277, pp. 136–146.
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*, ACM Press, New York, NY, USA, 2002, pp. 89–98.
20. C. Bádicá, A. Báditá, M. Ganzha, and M. Paprzycki, “Developing a Model Agent-based E-commerce System,” in *E-Service Intelligence - Methodologies, Technologies and Applications*, edited by J. Lu, Springer, Berlin, 2007, pp. 555–578.
21. Jena—a semantic framework for java, `http://jena.sourceforge.net`.
22. Fipa contract net protocol specification, `http://www.fipa.org/specs/fipa00029/SC00029H.html`.
23. J. Dodgson, M. Spackman, A. Pearman, and L. Phillips, *DTLR multi-criteria analysis manual*, UK: National Economic Research Associates (2001).
24. The insight research corporation, grid computing: A vertical market perspective 2006-2011, `http://www.insight-corp.com/reports/grid06.asp` (2006).
25. `http://www.enginframe.com/enginframe/demo/reservoir`.
26. `http://www.sun.com/service/sungrid/index.jsp`.
27. `http://www.globus.org/ogsa/`.
28. `http://www.doc.ic.ac.uk/~sjn5/GGF/ggf-rus-gsax-01.pdf`.
29. `http://www.doc.ic.ac.uk/~sjn5/GGF/gesa-wg.html`.
30. `http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF01/Hazlewood_SDSC.pdf`.
31. `http://www.gridbus.org/papers/gridbank.pdf`.
32. L. Catuogno, P. Faruolo, U. F. Petrillo, and I. Visconti, “Reliable Accounting in Grid Economic Transactions,” in *Book Grid and Cooperative Computing: GCC 2004 Workshops*, Springer Berlin-Heidelberg, LNCS, 2004, vol. 3252, pp. 514–521.
33. C. Bádicá, M. Ganzha, M. Gawinecki, P. Kobzdej, and M. Paprzycki, “Towards Trust Management in an Agent-based E-commerce System—Initial Considerations.,” in *Proceedings of the MISSI 2006 Conference*, edited by A. Zgrzywa, Wroclaw University of Technlogy Press, Poland, 2006, pp. 225–236.
34. M. Kinateder, E. Baschny, and K. Rothermel, “Towards a Generic Trust Model—Comparison of Various Trust Update Algorithms,” in *Trust Management*, Springer Berlin, 2005, pp. 177–192.
35. M. Kruszyk, M. Ganzha, M. Gawinecki, and M. Paprzycki, “Introducing Collaborative Filtering into an Agent-Based Travel Support System,” in *Proceedings of the IAT conference*, 2007, in press.