Scalable Computing: Practice and Experience
Volume 10, Number 4, pp. 397–411. http://www.scpe.org
ISSN 1895-1767


# MIRRORING INFORMATION WITHIN AN AGENT-TEAM-BASED INTELLIGENT GRID MIDDLEWARE; AN OVERVIEW AND DIRECTIONS FOR SYSTEM DEVELOPMENT

MARIA GANZHA, MARCIN PAPRZYCKI, MICHAL DROZDOWICZ*, MEHRDAD SENOBARI†, IVAN LIRKOV, SOFIYA IVANOVSKA‡, RICHARD OLEJNIK§, AND PAVEL TELEGIN¶

**Abstract.** This work concerns part of our project, devoted to the development of an agent-team-based Grid resource brokering and management system. Here, open issues that have to be addressed in the process, concern agent team preservation. In our earlier work it was suggested that this can be achieved through mirroring of key information. Here, we discuss in detail sources of useful information generated in the system (an agent team in particular) and consider which information should be mirrored, when and where, to increase long-term sustainability of an agent team.

**1. Introduction.** Our work is devoted to the development of an agent-team-based high-level intelligent Grid middleware. In our earlier work we have conceptualized a number of scenarios which require access to information that was generated within the team. For instance, it is assumed that when a team leader (the *LMaster* agent) receives (from an agent representing a *User*; the *LAgent*) a *Call for Proposals* (*CFP*) asking about conditions of executing a specific job, its response (an offer, or a rejection) should be based on its knowledge of the potential client (*User* that the *LAgent* represents), as well as of current market conditions (see, [20, 13]). Such knowledge is to be grounded in historical data collected during past interactions with potential client(s). For instance a series of unsuccessful bids for jobs may indicate that the suggested price is too high. Since we assume ( [25]) that a global Grid is a highly dynamic structure, in which nodes can disappear practically without any notice, it is important to assure that the team knowledge will not be lost when its leader crashes. Therefore, in [14, 26] we have suggested that support for long-term existence of agent teams may be achieved through utilization of information *mirroring*. Specifically, we have proposed that in each team an *LMirror* agent should be created, and its role should be to store a copy of all information necessary to prevent team disintegration in the case of crash of the *LMaster*. However, thus far we have not delved into any details as to what needs to be mirrored, when and how.

The aim of this paper is to describe sources of information that can be useful for an agent team in the context of its survival. Furthermore, we discuss when such information should be mirrored and how. To this effect we proceed as follows. In the next section we present a brief overview of the proposed system, as well as arguments for the need of information mirroring. We follow with presentation of sources of information that is to be collected. In each case we discuss when and where this information has to be persisted. Finally, we discuss what consequences does utilization of an outsourced data warehouse (to mirror team data) on procedures involved in recovering crashed *LMaster* and *LMirror* agents. Discussion includes a proposal for the structure of such data warehouse. Material presented here extends results introduced in [19].

**2. System overview.** To present a high-level overview of the system as well as to discuss two of its main functionalities: (i) *Worker* joining a team, and (ii) team accepting a job to be executed; we will utilize an AML Social Diagram ( [11]) in figure 2.1.

In the approach utilized in our system, agents work in teams. Each team is managed by its "leader," the *LMaster* agent. Agent teams utilize services of the *Client Information Center* (represented by the *CIC* agent), to advertise work they are ready to do and, possibly, *Workers* they are seeking. In addition to the *LMaster* and *Workers*, each team consists of an *LMirror* agent. This agent stores a copy of information necessary for the team to persist in case when the *LMaster* crashes.

When the *User* is seeking a team to execute its job, it specifies job constraints to its representative, the *LAgent*. The *LAgent* contacts the *CIC* to obtain the list of teams that can execute its job and utilizes trust information and the FIPA Contract Net protocol ( [5]) to either find a team that can do the job, or establish that such team cannot be found (within the specified constraints).

*Systems Research Institute Polish Academy of Science, Warsaw, Poland
†Tarbiat Modares University, Tehran, Iran
‡Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria
§University of Sciences and Technologies of Lille, Lille, France
¶SuperComputing Center Russian Academy of Sciences, Moscow, Russia

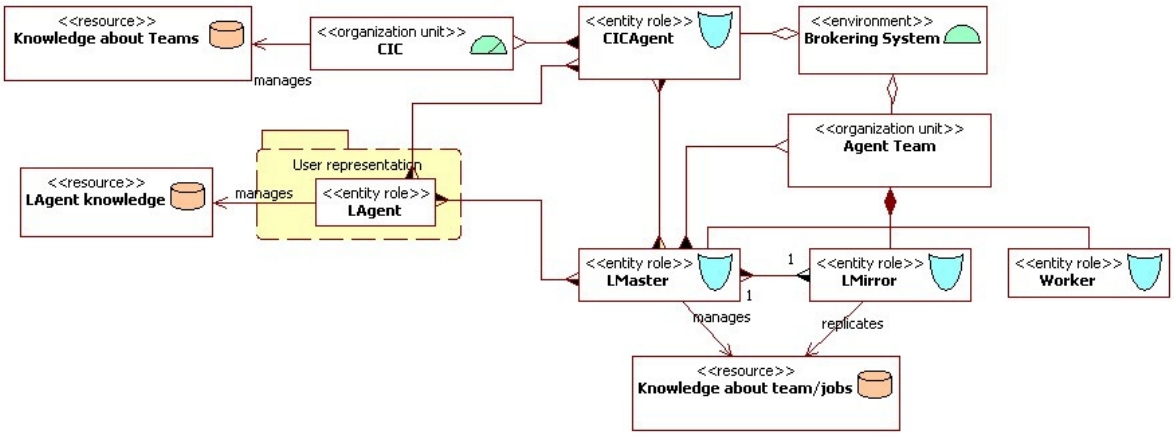


FIG. 2.1. *AML social diagram of the proposed system*

When the *User* would like to earn money by offering services of its computer(s), by becoming a *Worker* in a team, it specifies its conditions of joining (working for the team, e.g. the minimal "salary"). The *LAgent* interacts with the *CIC* to obtain a list of teams that are seeking *Workers* (satisfying characteristics of hardware and software it represents). Next, it utilizes trust information and the FIPA Contract Net protocol to establish if a team to join can be found. In both cases, trust information is a part of *LAgent*'s knowledge. At the same time, when responding to the CFP from *LAgents*, the *LMaster* utilizes the *Team knowledge* (more about *LMaster* responses can be found in [26, 20]).

**3. Need for information mirroring.** While issues involved in collecting knowledge by the *LAgent*, and utilizing it in decision making supporting the *User* are interesting, we focus our attention on processes taking place within the agent-team. This is especially so, since persisting information stored by the *LAgent* can be done by regularly backing-up the data (the same way as any *User* data should be backed-up).

Now, let us recall, that one of our key assumptions is that in the case of a *global Grid* any node can disappear practically without warning, and for an unspecified time (in the global Grid nodes are assumed to belong to individuals and thus, for all practical purposes, there are no guarantees of service). Obviously, this assumption has to be applied also to the *LMaster* agent (specifically, the node that it resides on). Let us now see what happens when the *LMaster* becomes unavailable and there is no mirroring in the team.

First, let us consider the case of a short-term disappearance of the *LMaster*. Since it is the "gateway agent" for the team, problems arising in this case involve, among others, contract negotiations (sending CFP to the *LMaster* or awaiting its response). Here, a short-term disappearance of the *LMaster* could result in loss of some potential contract (there would be no agent to send the CFP to, receive a response from, or confirm a contract with). Furthermore, since the *LMaster* is the gateway through which contracted jobs are sent to the team, and results sent back to *Users*, its short-term crash would result in some *Users* starting to believe that the team disintegrated. As a result, the reputation of the team would be damaged and some *Users* would not work with it again (these *Users* that were not able to contact the team during the *LMaster* crash). Recall that after the *CIC* delivers to the *User* the list of teams that potentially can complete its task, this list is pruned out of all teams that are not considered trustworthy.

Second, let us consider the fact that the *LMaster* is the only agent in the team that *Workers* know about (we assume that *Workers* do not know each other; at least such knowledge does not originate from the setup of our system). Thus, disappearance of the *LMaster* that lasts more than a few minutes, is likely to result in *Workers* not being able to send results of completed tasks and/or receive new tasks to work on (the only reprieve for the *LMaster* would be if all workers would execute long-lasting jobs and the *LMaster* would be back before any of them was completed). As a result *Workers* would consider their team (leadership) not trustworthy, leave it and not come back (see, [20]).

Finally, there is a scenario in which the *LMaster* crashes and "does not recover" (long-term disappearance, possibly with a serious data loss). Let us say that there was a fatal hard drive crash and there was no backup (or

the last backup was done long time before the crash). In this case the above mentioned problems are seriously magnified:

- since the *LMaster* is the only gateway between *Users* and the team, all jobs will be lost and thus the "User-reputation" of the team will be damaged very seriously; it is doubtful that *Users* would be willing to work with this *LMaster*, as a team leader, again
- the team will disintegrate, as *Workers* will have no *LMaster* to lead them; it is doubtful that any *Worker* affected by the crash would be willing to work with this *LMaster* again; furthermore, the *LMaster* after recovery will have no up-to-date knowledge about make-up of its team
- substantial part of other collected knowledge will be gone and thus the *LMaster*, even if it returns and tries to form a new team, will have to start from the scratch; but this would be very cumbersome indeed (while the team knowledge is gone, the competition / other teams will capture the "market share").

Thus, even a temporary disappearance of the *LMaster* would have serious consequences because of both *Users* and *Workers* considering it not worthy of trust. As a result, the team would lose *Workers* and stream of revenue (*Users* would not contract jobs with it) and would very likely disintegrate.

In response to these challenges we have decided to use a well-known strategy of data mirroring [1, 4, 23, 22, 33]. In [12, 25, 27] we have proposed that an *LMirror* agent should be created, with the role of keeping a copy of information needed to keep the team alive and competitive. Furthermore, in [28] we have specified procedures of restoring the *LMaster* and the *LMirror* in the case when either one of them crashes.

Obviously, we are well aware of the fact that instantaneous (or, even, almost instantaneous) crash of both the *LMaster* and the *LMirror* would have exactly the same consequences as crash of the *LMaster* that did not have an *LMirror* and data backup (would result in destruction of the team). However, our approach is not focused on developing a completely bullet-proof system, but our goal is to create a reasonably resilient infrastructure. In this context observe that, as the time passes, it can be assumed that, in best teams, roles of the *LMaster* and the *LMirror* will be played by more-and-more reliable machines. Specifically, when the *LMaster* selects the *LMirror*, it will pick for this job the "best possible" *Worker*. It will do so by utilizing the reliability information it has accumulated thus far (see, [20]). When this *LMirror* becomes the next *LMaster* it will also select the best *LMirror* available at this time. In this way the overall strength and reliability of "team leaders" should continue to increase. Obviously, from what we described here follows that as the time passes weakest teams may be naturally eliminated. Specifically, teams with unreliable *LMasters* and *LMirrors* will either fall apart when both these agents crash at almost the same time, or due to repeated temporary disabilities, lose chances of winning new contracts or fulfilling existing ones. As a consequence such teams will lose *Users*' trust, while their *Workers* will not come back (to work for a "bad team"); and this will result in diminishing funding and, finally, team disappearance.

In our earlier work we have focused our attention on the structure of the team, and on the way that lost leader agents are replaced (and the structure of the team restored). Now, we will consider three key issues: (1) what information should be mirrored, (2) when should it be mirrored, and (3) where should it be mirrored.

## 4. What should be mirrored, when and where.

**4.1. Team member data.** For any team, the most crucial is information about its members. Each time a new member joins the team, the *LMaster* obtains the following information (note that all information is "managed" utilizing an extended Grid ontology; see [17]):

1. *ID* of the *LAgent* that is joining the team (*Worker* name). Since we have assumed that each agent utilizing our system has to be registered within it (this function is one of the roles of the *CIC* infrastructure), it is possible to check the registration of the potential *Worker*, before accepting it into the team (see, [15] for more details). This can be treated as a mini-security measure; one that can be extended as the system develops.
2. *Available resources.* While in [13] we have proposed a very minimalistic ontology of resources, we have known all along that a more detailed one is needed. More recently, in [30] we have presented an overview of Grid and agent-Grid ontologies and came to the conclusion that an extended (and somewhat modified) CoreGrid ontology [37] will be the base ontology for our system. Thus, this ontology will be also used to formulate an offer to join the team. Specifically, in the CFP the *LAgent* will include information about resources offered to the team. At this point, as we are considering resources in terms of computing nodes (consisting of hardware and software), the top entity of the resource description will be the *ComputingComponent* class and its subclasses—*ComputingElement* and *WorkerNode*. This

will enable *Worker* agents to represent either a single computing resource (*WorkerNode*) or a set of computing nodes, managed by some low level resource broker (*ComputingElement*, containing a set of *WorkerNodes*). Instances of both of these classes can have their hardware and software capabilities described by properties such as (this list is not exhaustive):

- *hasCPU*—related to the CPU class and describing the processor in terms such as: *platform*, *vendor*, *model*, *clockSpeed*, *coreCount*, *l1CacheSize*, *l2CacheSize*, etc.
- *hasMemory*—linking to the *Memory* concept described by the *Size* property.
- *hasStorage*—related to the *StorageSpace* class described further by the *StorageInterface* and the *FileSystem* concepts.
- *installedSoftware*—describing any software components of the node (the *Software* class) such as the operating system, application environment and installed libraries.
- *runningService*—any services running on the resource (e.g. OpenPBS broker running on the *ComputingElement* node).

This information about the new *Worker* will be then stored by the *LMaster* and used, among others, for (1) contract negotiations, and (2) job scheduling (see also [31]).

3. *Details of the contract.* As can be found in [28], we have suggested that the contract proposal should contain: (a) specification of the time of availability (e.g. every night from 1AM till 6AM), (b) length of contract (e.g. 2 weeks). However, results obtained by the MiG project (see, for instance papers available at [3]) illustrate that more complex sets of informations can be involved in both contract negotiations as well as in the contract (e.g. a contract with max payment value). Obviously, in the response the proposed payment for the *Worker* (and its structure) is to be provided. These details (regardless of their final complexity, resulting from extended analysis of possible economical models) constitute the description of the contract between the *LMaster* representing the team and the *Worker*.
   The exact way of modeling these concepts in our Grid ontology is in progress, however currently, they are described using the *WorkerContract* class along with the following properties:
   - *revenuePerHour*—numerical value describing the price the *Worker* is paid for each hour of its readiness to work.
   - *availability*—a multivalued property containing times of days during a week during which the *Worker* is available for job processing. The domain of this property is a list of *AvailabilityPeriod* class instances, described by properties: *startDayOfWeek*, *startTimeOfDay*, *endDayOfWeek*, *endTimeOfDay*.
   - *contractStartTime*—date and time when the contract came into effect.
   - *contractLength*—the length of the contract in days.

Out of these three items, one has to be mirrored immediately—the *ID* of the *Worker*. Without it it will be impossible to contact it in the future. The remaining two could be recovered from the *Worker*.

As far as the resources of the *Worker* are concerned, we have also to take into account the fact that in the system under consideration it is expected to that the *LAgent* will "remain the same" (its ID does not change) for a long time, while the resources it represent may change. Let us consider two scenarios. First, the *LAgent* represents one or more "home PC's." Here, machines can be added to the pool, removed from the pool, and/or upgraded. Obviously, such changes are not likely to happen each time the *LAgent* interacts with the team. However, it is even possible that one of home machines will be relatively regularly added to, and removed from, the pool of available resources. Second, the *LAgent* represents a Grid (e.g. an ADAJ based desktop Grid; see [15] for more details). This case can be expected to be even more dynamic as it is possible that each time the *LAgent* joins the team it will represent a Grid that is configured out of a different set of machines. Therefore, information about resources represented by a given *LAgent* is not static and needs to be stored each time it joins the team. Furthermore, it should now be obvious that in the case of long-term contracts it may be necessary to regularly update information about available resources (particularly in the case of an *LAgent* representing a Grid or a Cloud). Therefore, as a result of possibility of such changes a different type of a contract may need to be negotiated. This latter issue we will, however, omit as being out of scope of this paper.

Let us now consider the question of preservation of details of the contract. When designing our system we have assumed that participating agents will not only be benevolent, but also their behaviors will be "team-supporting" (pro-social). In other words, all participating agents will try, to the best of their abilities, to cooperate with others. However, such assumption is rather unrealistic when real-world situations are to be considered. Moreover, in the agent literature it is quite often the case that non-collaborating, anti-social

agent behaviors are considered (see, among others, work of D. Grosu, for instance [21]; or H. Hexmoore, for instance [36]). Taking this into account one can envision a malevolent *Worker* who tries to take advantage of crashing of the *LMaster* by providing the *LMirror* with details of the contract that have been modified to its advantage. Therefore, one of the easy ways of preventing such misbehavior of the *Worker*, is to mirror details of all contracts. In other words, information about the contract of the new/returning *Worker* should be mirrored as soon as the contract is signed (the FIPA Contract Net Protocol is completed).

Summarizing, information about ID of the worker (item (1), above) has to be mirrored immediately; information about details of the contract (item (3), above) should be mirrored immediately to prevent potential fraud; while information of resources represented by a given *Worker* (item (2), above) does not need to be mirrored. However, since information about items (1) and (3) is to be mirrored, it seems reasonable to mirror also information describing available resources (item (2)). In this way crashing of the *LMaster* will not disturb *Worker*s with unnecessary exchanges of messages with the new *LMaster*; preserving resources of both sides. Note that the total amount of information mirrored here is proportional to the number of *Workers* in the team (modulo the number of machines that each one of them represent) and thus should not be very large. Furthermore, change in the total size of mirrored data can be estimated in the case of planned team expansion or reduction (e.g. it is easy to establish how much disk space will be required if another *Worker* representing $N$ machines is to join or leave the team).

**4.2. Job contracts and their execution.** In [14, 13, 12] we have specified a very limited set of parameters describing job contracts. In the CFP, the potential *User* could specify the following parameters: (a) job start time, (b) job end time, and (c) resource requirements (where resources have been limited to available hardware—matching resource information obtained from each *Worker*; see the previous section). In the response, the price for executing that job was proposed by the *LMaster*. As noted above, when introducing to our system a full-blown ontology of the Grid, additional job constraints are going to be added. However, this does not affect considerations presented here. Obviously, as soon as the contract is signed (FIPA Contract Net Protocol is completed), all data concerning it has to be mirrored. Information about signed contracts is crucial to the continued existence of the team, as each signed contract means income for the team. Furthermore, losing information about a contract would mean that it would either not be completed, or results would have no *User* to be sent to (the latter would happen in the case, when information was lost while the task was already being executed by one of the *Workers*). In both cases, the team would rapidly lose its reputation and potential future contracts ( [20]). Furthermore, as noted above, as soon as the assumption about pro-social, benevolent attitude of all agents in the system is relaxed, one of the ways to protect interest of all parties is to mirror all contract information.

Signing a contract leads to an issue that is interesting in its own right: what should happen with job-related files? As it is obvious for anyone working with Grids (and illustrated in [35]), a typical job description involves a file (or multiple files, when job orchestration is concerned) that contains the code, and one or more data files. The first possibility would be to request files from the *LAgent* at the time when the job is "about to start." This approach would reduce the burden on the team. When the job was to start (either because of contract stipulation—the *job start time* condition—or because time to execute a given job has come) the *LMaster* would contact the *LAgent* with a request to send all necessary files. The disadvantage of this solution is that there is no guarantee (and in this case there should be none) that the *LAgent* is going to be available at the time when the files are to be requested. From the point of view of the *LAgent* its job is to negotiate the contract, deliver the files to the team that is to complete the task, and receive the results. In the meantime the *LAgent* can be off-line (computer it resides on can even be switched off). This is particularly the case when the *job end time* condition is a part of the contract (it will come back to life when it is time to pick up the results). This brief analysis points out to the fact that we have to assume that completing the contract is primarily the responsibility of the agent team. As soon as the *LMaster* signs the contract, the team takes responsibility for completing the task (note that the payment for completing the task can be assured using appropriate proxy mechanisms; [20]). Therefore, solution that expects files defining the task to be transferred when the job is about to start is not acceptable. Such files have to be transferred to the *LMaster* immediately after the contract is signed.

With this in mind, let us consider possible actions of the *LMaster* that has received all files pertinent to a newly contracted job. Naturally, the *LMaster* could immediately establish which *Worker* is to execute the job and initiate job staging, by sending all files to that *Worker*. Unfortunately this is the situation when, our fundamental assumption—about highly dynamic nature of nodes in the global Grid comes to play. Determining,

well ahead of job start time, which *Worker* is going to execute it, seems to be against the very nature of utilization of agent systems. Here, the robustness and efficiency of such systems comes from their flexibility (e.g. ability to negotiate and re-negotiate "job contracts;" as described, for instance, in [32, 8, 7]). Therefore, it seems that it would be much better to stage jobs according to the *actual* state of the system at the the time of job start (rather than on the basis of the predicted state of the system). This approach should allow to avoid influence of unexpected factors, such as, for instance failure of some *Workers*. This means that data files should not be immediately transfered from the *LMaster* to the *Worker*. Combining these two observations we realize that all files necessary to complete the job have to be held by the *LMaster* until the start of job execution. This means, in turn, that these files have to be mirrored.

As soon as the job is completed, its results have to be sent back to the *User*. Now, let us recall that the only agent that knows the *User* is the *LMaster* and thus the *LMaster* has to send them back. Here, there exist two possible mirroring options. First, while the *LMaster* keeps a copy of results, the second copy is kept by the *Worker*; until it receives a confirmation from the *LMaster* that they were successfully delivered to the *User*. In this way we, again, have two copies of sensitive material available in the system. Furthermore, note that the overall reliability of the *Worker* is assumed to be smaller than that of the *LMaster* and the *LMirror* (the latter two agents have been selected on the basis of much more stringent criteria, and their overall capabilities and their reliability should improve as the team continues to exist). Therefore, the situation in which important data is not "properly" mirrored should not be sustained for an extended time. However, note that the *Worker* can end its contract and not come back to the team. In this case, if the *LMaster* crashes, results could be lost before they are delivered to the *User*. To deal with such situation, if data cannot be delivered immediately to the *User* (e.g. an appropriate *LAgent* is not available), results have to be mirrored as any other sensitive data and the *Worker* should be released from the duty of keeping a copy.

Information about job completion and successful delivery of results has to be stored. It is going to be useful, first, for utilization in job scheduling (see, for instance, [31] and references collected there). Second, for contract disputes, and third, for considerations concerning trust. Interestingly, only the fact that the job has been completed and its results successfully delivered to the customer, has to be mirrored immediately (this information describes the current status of the team, which has to be kept up to date). Information needed for future job scheduling does not have to be mirrored immediately. Losing some data about execution times of some jobs (due to the crash of *LMaster*) does not pose a threat to the existence of the team. Similarly, some information related to trust assessment of a *Worker* may be lost without damaging the team. Therefore, these two item-sets can be mirrored in a digested form in pre-specified time intervals (see subsequent sections).

**4.3. Trust information.** Thus far we have dealt primarily with information that has to be mirrored immediately, now we devote our attention to the remaining data generated in the system. In [20] we have considered issues related to trust in relationships between *Users* and *LMasters* (teams) and between *LMasters* and their *Workers*. We have shown that, utilizing a proxy environment (similar, for instance, to PayPal [2]) it is possible to assure that payment for a completed job is always delivered. Therefore, for the purpose of this paper, we are interested only in relationships between the *LMaster* and its *Workers*. Obviously, since we are interested in information mirroring within agent teams, the way that trust information is to be stored by *Workers* is not considered.

Following the discussion presented in [20] we assume that, for the time being at least, contract between *Worker* and the team (the *LMaster*) is based on paying the *Worker* for availability (not for actual work done); i. e. *Worker* contracted for 6 hours every night will be paid for this many hours and will be expected to be available within that time-frame to do work, if called. It was also stipulated that, at least for some contracts, *Worker* will have a right to not to be available for a limited number of times during the contracted time. Here, we can use the "pinging-mechanism" described in [12] to monitor *Worker* availability. Under these assumptions, in [20] we have proposed that for each *Worker* we will collect information how many times (a) it *fulfilled the contract* ($\#fc$)—was available all the time when it was expected to be available (if in the contract it was stated that it can be missing for a certain number of times, this is exactly the number of times it was missing); (b) violated the contract ($\#vc$)—was not available when it was supposed to be; and (c) did more than contract required ($\#ac$)—could have been unavailable for a specific number of times during a contract, while was available all the time. This combined data will be stored, for each *Worker*, as a quadruple $(n, \#fc, \#vc, \#ac)$, where $n$ is the total number of contracts. While this information is collected in a cumulative form (and thus is of size proportional to the number of *Workers* in the team), we have also access to additional information

that can be used for assessment of trust / reliability of a *Worker*. First, detailed information about results of each "pinging-procedure," and second, detailed information about completion (or not) of each assigned job (e.g. how much time a given job took, was it completed on time, before time, or delayed, etc.). Note that the results of the "pinging-procedure" can be used also to adjust time within which response from a given agent is expected (in this way system will be able to adapt its behavior to the network conditions of each *Worker*; see, also [28]). Since the latter data is not proportional to the number of *Workers* in the team, but to the number of "pinging-procedures" and processed jobs, we consider it separately in section 4.4.

Obviously, the trust-related information has to be mirrored. However, it is easy to realize that loss of some of this information is not catastrophic for the team; e.g. not updating immediately number of fulfilled contracts for a specific *Worker* will result in proceeding with the old trust assessment (note that the basic data is cumulative). Therefore, it is enough to update information about trust related issues in a digested format at predefined time-intervals. Frequency of updates depends on the nature of jobs that a given team is executing and on the nature of *Worker* contracts. If a team is contracting long-lasting jobs and its members have long-term contracts, then update of trust information can occur much less frequently than in the case when large numbers of short jobs are completed by a team consisting of *Workers* with short-term contracts. We believe that a good approximation of the right time to send an update of trust information to the *LMirror* is when, in average, each *Worker* has completed one contract. Note that updates of information can be sent at any time, which allows the *LMaster* to adaptively adjust time between updates depending on the nature of jobs and contracts currently existing in its team.

**4.4. High volume data collection and storage.** Most information considered thus far within the system was relatively small in volume—on order of the size of the team, or of the number of currently contracted jobs (note that in the latter case files related to some jobs may be large, but this is unavoidable). There are, however, important data sets that grow over time and can become very large. First, data generated during negotiations between the *LMaster* and *Users* who would like either to contract job execution, or join the team. This data is to be used to establish market conditions and introduce team behavior adaptivity to the system (see, also [12, 20]). Second, information about execution of completed jobs (e.g. hardware used, software required, execution time, *Worker* completing the job, etc.). This data can be used to apply advanced scheduling techniques (for more details, see [31], and references collected there). Third, detailed information about behavior of each worker (e.g. detailed history of execution of each job, results of "pinging-procedures," etc.). This information is crucial to build a realistic and comprehensive economy-based job scheduling model (which includes also trust and reliability related considerations).

Let us first consider contract negotiations. In [34, 6, 9, 10] a large variety of negotiation mechanisms that can be applied in a Grid have been described. However, in the initial implementation of our system, for both types of contract negotiations (job execution, or joining a team), we have decided to utilize the FIPA Contract Net Protocol ( [5]). The primary reason was that while allowing for "calls for proposals" with practically unlimited variety of constraints, it generates the contract (or establishes impossibility of an agreement) within a single round of negotiations. This also reduces the total amount of information that needs to be stored for further use (compare, for instance, with the amount of data generated during any of the multi-round auction mechanisms; see [16]). Therefore, in our system, information about a single job contract negotiation would consists of:

- *Content of the CFP*—containing, among others, information what hardware and/or software was sought; job start time (if specified); how much time was to be contracted / or was it an open-ended contract?
- *Content of the response*—what was the price proposed by the *LMaster*, as well as other details of the proposed contract; e.g. the proposed hardware (when the CFP specifies the minimum requirements, while the response proposes the actual configuration that the task will be executed on), etc.
- *Result of the negotiation*—was it a success, or not?

Note that, with a rich vocabulary provided by the extended Grid ontology, it will be possible to reconsider the utility of multi-round Service Level Agreement negotiations.

In the case of negotiation concerning a *Worker* joining the team, at least the following information becomes available (introduction of a full-blown Grid ontology will allow utilization of more complex negotiation scenarios):

- *Content of the CFP*—including information about available hardware and software, length of contract and conditions of availability,

- *Content of the response*—the proposed price
- *Result of the negotiation*—did the *Worker* join the team, or not

Note that, regardless of the particular form of negotiation used, amount of available data will be large. Specifically, it will be proportional to the total number of contract negotiations that a given team participated in. Furthermore, this dataset will be constantly increasing in volume.

Interestingly, in all cases, data collection is resilient to loss of some information. In other words, losing some of the data may decrease competitiveness of the team (e.g. when performing data mining to find the current "value" of various resources that the team currently consists off, the result may be somewhat incorrect due to the fact that some data was lost), but the process is characterized by graceful degradation. Therefore, it is possible to perform updates in a digested fashion at times of reduced utilization of the system, while risking that some data will be lost if the *LMaster* crashes in the meantime.

It should be now obvious that constantly increasing volume of data makes mirroring very costly. We can point to at least three important issues that need to be taken into account.

1. The total amount of data that has to be regularly passed from the *LMaster* to the *LMirror* is not small. This is the case even when digested information is transferred in a compressed form. Here, we deal with a constant stream of data, size of which depends on the size of the team and the level of activity within it.
2. Constantly increasing volume of data has to be stored by *both* the *LMaster* and the *LMirror*. Even assuming that over time capabilities of both managerial agents improve, and taking into account that storage is cheap and plentiful, collecting data generated during management of a large and active team that exists for an extended period of time, may be prohibitive for most home PC's. This will, in turn, introduce a new stratification into the Grid. Its *Users* will be divided into those who can and those who cannot manage the team due to the storage restrictions of their computers.
3. The time of recovery of a crashed *LMirror*, and of the second step of replacing a crashed *LMaster* (in the first step the new *LMAster* replaces the *LMirror* on the same machine, and thus no data transfer is needed) is going to grow with the data size. It is important to realize that (re)creation of a new *LMirror* involves copying an entire set of mirrored data to the new location. For large, active teams that exist for a long time, such data set can be very large. As a result transfer involving ACL messages is not feasible. The only reasonable solution seems to be a direct data transfer between computers that host both agents. However, even this process can take considerable amount of time, if it is performed over the Internet. Now, let us take into account that, as suggested in [27], recovery of a crashed managerial agent stops the other managerial agent from doing anything else (to re-establish security of existence of the team; brought about by data mirroring). This means that as the time passes each crash of the managerial agent takes longer to recover from (more data to be mirrored) and, for all practical purposes, stops the team operation for an ever increasing time. Therefore, such approach seems infeasible in a long run.

In this context note that, to reduce potential impact of increasing volume of collected data, it would be possible to store only the newest information (e.g. a "sliding window" consisting of $K$ most current data items). However, the primary reason to collect a "complete set of historical data" is to be able to apply data mining techniques to extract useful information out of it. Decreasing the amount of collected data, by including only the most current items, could reduce effectiveness of data mining and thus contradict the reason to collect this data in the first place. Therefore, it seems to be in the best interest of the team to preserve as much of historical data as possible.

Considering this, on the basis of the current trend of IT infrastructure—outsourcing and specialization, we propose that an outsourced data warehouse will be utilized for team data storage. Specifically, within our system, data warehousing could be outsourced to a team that *specializes* in data storage. This is similar to, for instance, a popular e-commerce scenario, where merchants contract software companies to design, implement and run infrastructure of multi-front e-stores, while software companies contract storage companies to actually store the data. Note that while we use the name team; it is quite possible that actually this will be a single entity (as in the case of e-commerce solutions) that will be interfaced with our teams through the agent infrastructure. In other words, the front end of the data warehouse could be one or more agents, but there could be no actual agent team working "behind it."

Obviously, it would be also possible to hire *Workers* with the right infrastructure to facilitate data warehousing within the team itself. However, this solution has a number of potential problems. First, recall the

assumption that each node (*Worker*) can disappear without advanced warning. This assumption would apply also to the *Worker*(s) responsible for team data storage. As a result, potential problems that lead us to propose data mirroring in the first place, would remain unsolved. Second, idea of internal data warehousing goes against the current trend in IT, which is to outsource anything that is not the core functionality (as long as it can be outsourced, e.g. currently it is unfeasible that a bank would outsource its IT operations). The reasons are the same, as data warehousing is not expected to be the core business of most teams. Data warehousing requires a specific infrastructure (e.g. a server farm) and procedures to assure data persistence and security. Therefore, on the one hand, it is better to create a facility that specializes in this type of operation and sells its services to others; on the other it is better to buy service that assures quality of data preservation.

Finally, let us observe one more benefit of utilization of an external storage facility for system data. One of the important features of the system (see, also Figure 2.1) is that all collected data will be used for knowledge extraction. The question thus arises, how will this be done? For instance, assuming that data is stored by the *LMaster* and the *LMirror*, which dataset will be used for data mining? Since the *LMaster* is the gateway and responsible for all contacts with *Users*, it seems that this is not the right node to stage the second agent, running data mining algorithms. At the same time, the *LMirror* may not have the most current set of data. However, since it has "much less to do," this node is the one that could be easily used for data mining (possibly, performed by a separate data mining agent). However, the situation is much simpler in the case of an outsourced data storage. Here the data mining agent can be instantiated within this (external) infrastructure, or run as one of *Workers* within the team (if running it within the storage facility is prohibited). In this way we can clearly separate the knowledge management function of the system and devote appropriate resources to it.

**4.5. Structure of the data warehouse.** Let us now discuss how a data warehouse that could be used in our system to store the above indicated data, may look like. What we present below is a draft design of a data warehouse that could be used for storing historical data for efficient analyzing and reporting. Obviously, additional storage would be needed for preserving short term / dynamic data described in sections 4.1, 4.2. However, in this case it sill remains an open question if such data should be stored only in the warehouse, or if it should be mirrored also by the two managerial agents (see, section 4.6. Data warehouse design presented here is based on principles laid out in [24], and is built around the star table schema. Note that some details of the proposed design may change as the system evolves (e.g. when new types of complex negotiations are added). However, the general schema, which is based on the extended Grid ontology, should remain unchanged.

**4.5.1. Worker negotiation.** Let us start from storage of information about negotiations between the *LMaster* and the *Worker* wanting to join the team. This data is going to be stored in the *Worker Negotiation Fact* table (see Figure 4.1.

Each row of the table will correspond to a single negotiation and is going to be identified by the following dimensions:

- *User*. Identifies the *User* of the system. In this preliminary draft it contains only the *Name* of the *User*.
- *Worker*. Describes the identity (i. e. the *ID* of the *LAgent*), configuration and proposed contract conditions of the *Worker* requesting to join the team.
- *Software Group*. A multivalued dimension that describes the software configuration that is going to be contributed by the *Worker*. Each *Software Group* can be composed of many rows of the *Software Dimension* table, where each of them describes a single software component.
- *Team*. The dimension identifying the team. So far it does not contain any additional meaningful information.
- *Date*. This is a typical date dimension, as described in [24], which enables easy analysis and reporting of aggregated time series data.

Measures of the *Worker Negotiation Fact* table consist of data about money offered as compensation for the *Worker* and the information whether the *Worker* was accepted. These can be used, for example, to estimate the market value of capabilities offered by the *Worker* (e.g. repeated rejections could mean that the offered compensation is too low).

**4.5.2. Job negotiation.** The *Job Negotiation Fact* table gathers information about the job submission negotiations. The diagram of this fact table along with its dimensions can be found in Figure 4.2.

Similarly to the case of the *Worker Negotiation Fact* table, each row corresponds to a single negotiation, and is identified using the following dimensions:

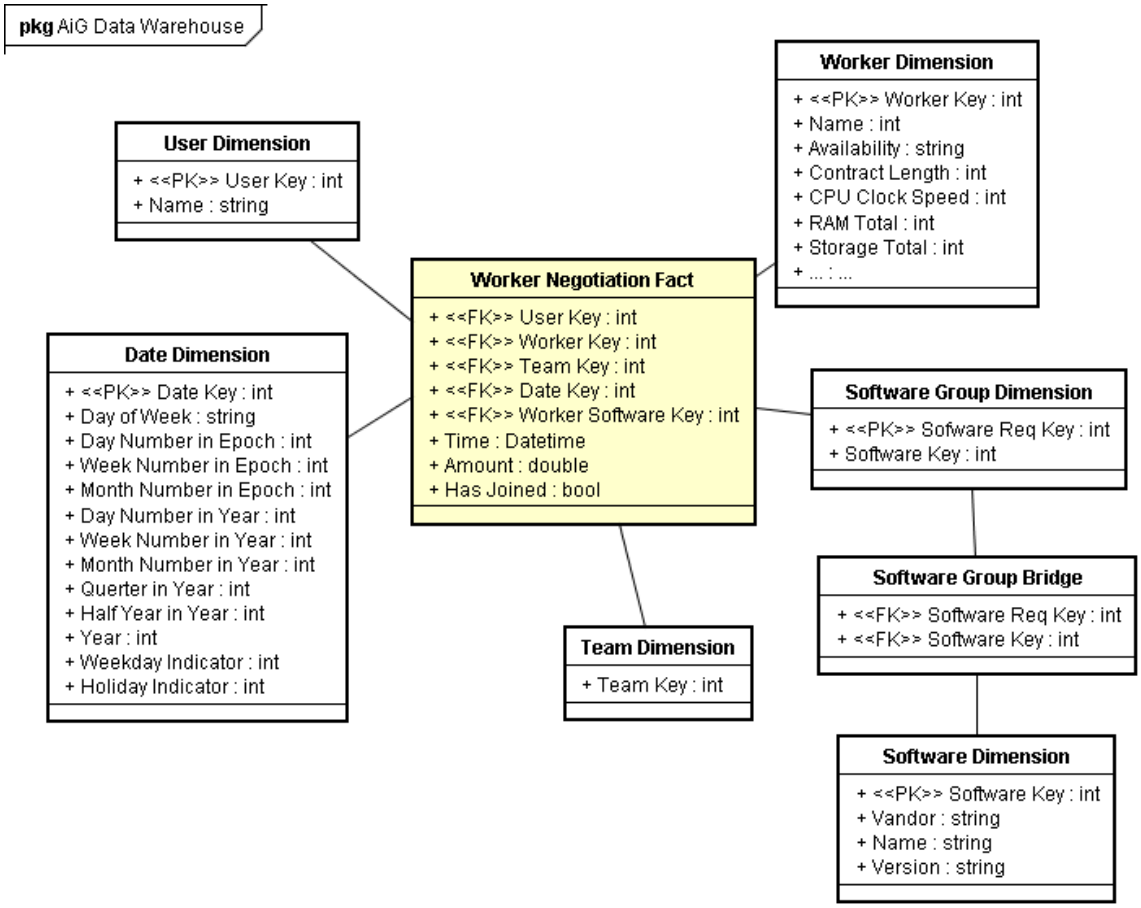


FIG. 4.1. *Worker Negotiation Fact Table*

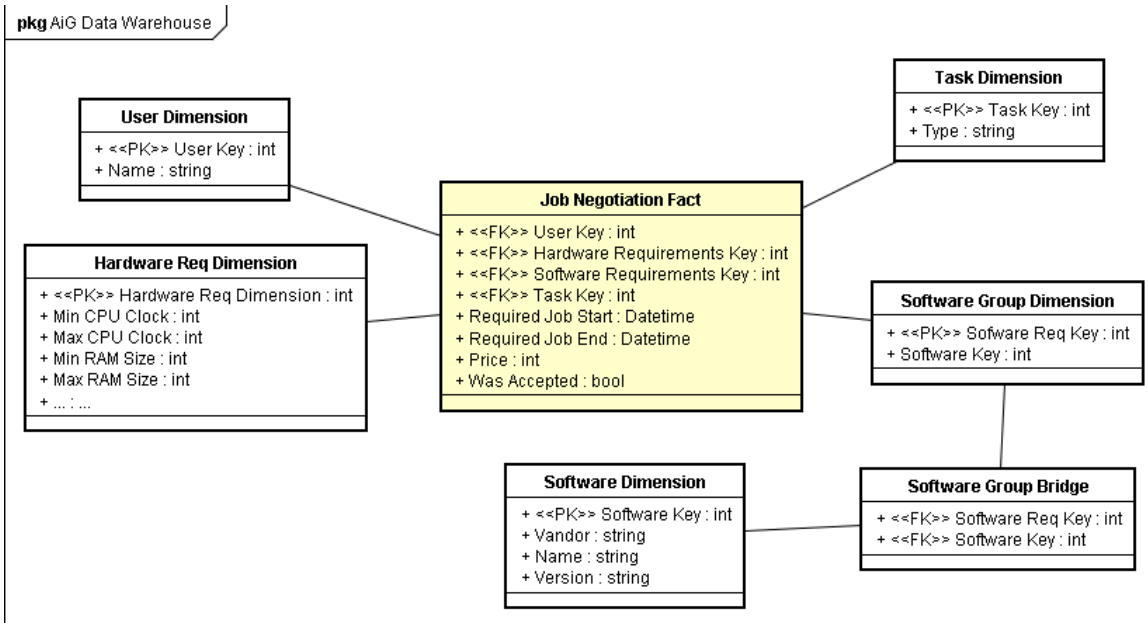


FIG. 4.2. *Job Negotiation Fact Table*

- *User*. Identifies *User* who would like to have a job executed.
- *Hardware Requirements*. This dimension describes the hardware requirements as specified by the *User*. Properties of this dimension are specified using a subset of concepts from the ontology, but flattened out and used as constraints.
- *Required Software*. Links to the *Software Group* dimension that describes software required to execute the job.
- *Task*. Describes the task (job) that the *User* would like to have executed. Details of how a task can be described or categorized have not yet been fully conceptualized; therefore, this dimension is currently empty.

Measures currently stored in the table are: the price proposed by the *LMaster* for executing the job and a boolean value specifying whether the *User* accepted the proposed conditions. Similarly as above, these dimensions may be extended in case of more complex negotiations. Furthermore, they are to be used, among others, to evaluate market conditions and will play key role in the economic model that the system is based on.

**4.5.3. Task execution.** Information about every task executed by the team is stored in the *Task Execution Fact* table (see Figure 4.3).

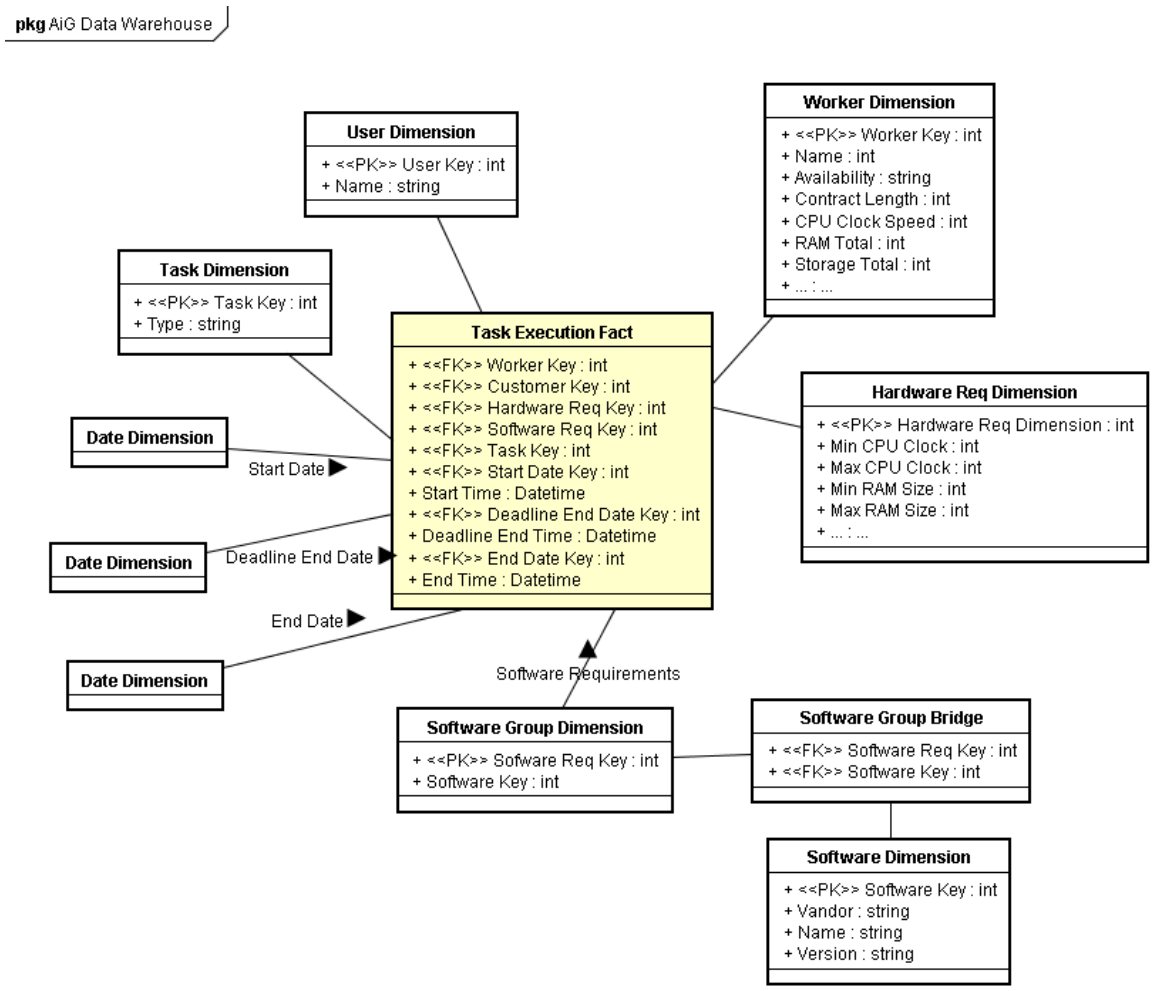


FIG. 4.3. *Task Execution Fact Table*

The table is described using the following dimensions:

- *Worker*. The worker that performed the task.
- *Customer*. The *User* that submitted the job.
- *Hardware Requirements*. Hardware that was required to perform the task.

- *Software Requirements*. Software necessary to complete the task.
- *Task*. The description of the executed task.
- *Start Date*. Link to the *Date* dimension, specifying the start of the task execution.
- *Deadline End Date*. Link to the *Date* dimension, specifying the deadline, as it was negotiated with the customer.
- *End Date*. The actual end of task execution.

For all of the *Date* dimensions (start, deadline, and actual end) there are also separate *Date* and *Time* properties to facilitate precise measurements. These measurements can be used, among others for task scheduling (based on historical data, see [31]).

**4.5.4. Worker responsiveness.** The information about the "responsiveness" of *Workers* within the team is stored in the *Worker Responsiveness Fact* table. It contains all data necessary to analyze the actual availability of the *Worker*, and conformance to the Service Level Agreement between the *Worker* and the *LMaster*. Recall, that in the current design of our system it is assumed that *Workers* are paid for being available at specific times. The design of this table is rather simple and is presented in Figure 4.4.

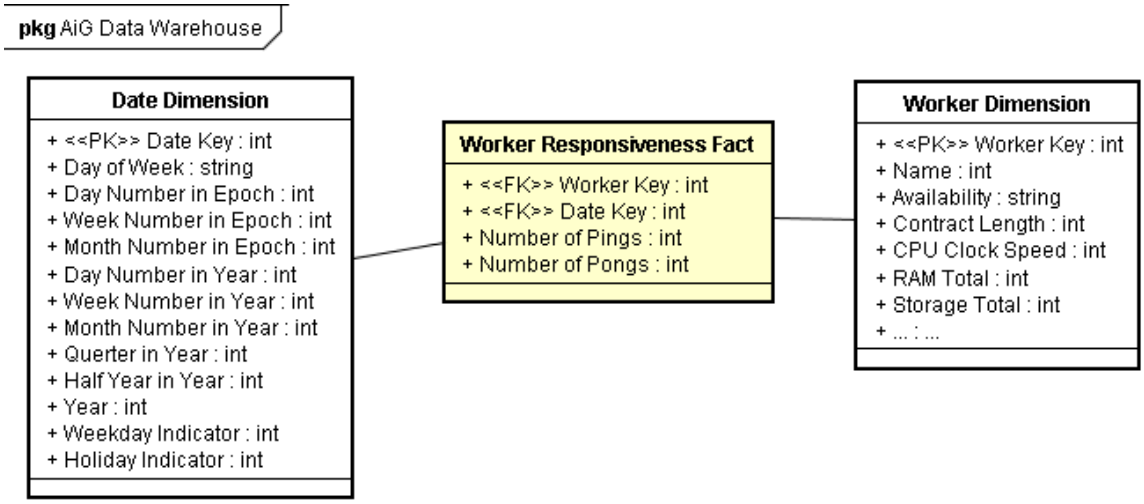


FIG. 4.4. *Worker Responsiveness Fact Table*

The table is identified by two dimensions:

- *Worker*. Identifies the *Worker* that the 'responsiveness" data concerns.
- *Date*. Specifies dates for which the measures apply.

Collected measures that can be used to analyze *Worker* "responsiveness" are:

- *Number of pings*. The number of messages checking the availability of the *Worker* sent by the *LMaster*.
- *Number of pongs*. The number of responses confirming the availability of the *Worker*.

It is likely that the collected measures are going to be extended, but they will remain stored in the same table.

**4.6. Consequences of utilization of outsourced data storage.** Let us now observe that utilization of a data warehouse allows us to redefine "roles" of and interactions between the *LMaster* and the *LMirror*. Thus far the *LMaster* and the *LMirror* were assumed to store and regularly synchronize **all** data pertinent to the long-term existence and success of the team. In the new design, the *LMaster* stores all such information in the contracted storage facility and thus the *LMirror* does not store any data at all. However, it should be clear that this does not mean that the *LMaster* does not store any data. It would be highly inefficient to query the external data warehouse every time some information is needed (for instance, to send a message to any/all team member(s)). Therefore we have to divide available information into three categories. (a) Information copy of which is definitely stored by the *LMaster*, (b) kept a copy if possible (depending on its capabilities), and (c) information stored in the warehouse only.

In category (a) we include current information concerning: list of team members, their resource specification, contract details (both job contracts and *Worker* contracts). In category (c) we include all high volume data

discussed in section 4.4. The remaining information discussed here, such as, for example, cumulative trust information, market value of specific hardware, etc., may be stored by the *LMaster*, but can be also accessed from the data warehouse.

Let us now come back to the *LMirror*. It does not store any data, but has procedures for: (a) checking existence of the *LMaster*, and (b) becoming an *LMaster* in the case when it crashes. Note that this simplifies the overall situation of the *LMirror*. Thus far one of the important issues was, should the *LMirror* work in a similar way as any other *Worker* agent. It was assumed that the *LMaster* will not work as a *Worker*, but will be 100% involved in team management. It will also be paid for its efforts from the overhead imposed on all contracts (see, [29] for more details). However, in the case of the *LMirror* the situation was not this obvious. On the one hand, it had to store team data and regularly check that the *LMaster* is still alive. On the other hand, since these actions require much less effort that these of the *LMaster*, it could perform some tasks as other *Workers* do. However, as a *Worker* it would not be as efficient as the others. Recall, for instance, that it was to process digested and compressed packages containing information about negotiations that took place recently. This being the case the *LMirror* would have to stop processing the *User* request and *immediately* take care of mirroring-related operations (since, assumptions behind our system setup specify that these operations have precedence over regular work). Let us also recall the suggestion made above, that one of the functions of the *LMirror* could be mining the team data; as an add-on function that could be performed instead of being a regular *Worker*. Regardless of the final solution used, this lack of clarity as to functionalities and payment (how much should the *LMirror* be paid for its services in each of the above described approaches?) would make the design of the system unclear and maintenance more difficult. This particularly concerns the economic model behind the system that would become more / unnecessarily complicated.

Situation becomes much clearer when the outsourced data warehouse-based solution is used. Since the *LMirror* does not have to store data it can work as all other *Workers* (the only mirroring-related procedure will be checking existence of the *LMaster*—using the pinging-procedure, see [28]). Obviously, if the *LMaster* crashes, the *LMirror* will have to undertake emergency procedures and become the *LMaster* and re-create the *LMirror*. However, this being an emergency situation requires special measures (e.g. involving dealing with the job it was processing). As suggested above, mining data stored in the warehouse can be delegated to a special data mining agent. In the case that this agent cannot move to the node where the data is stored, this could be a natural work for the *LMirror*, as in this case the data mining task it is executing can be stopped at any time. Furthermore, the economic aspect of the system can be simplified, as the *LMirror* can be paid as any *Worker*.

**4.6.1. Restoration of managerial agents.** Let us now briefly summarize steps that take place when a crashed managerial agent is going to be re-created in the scenario when team data is persisted in the external data warehouse. First, let us consider re-creation of a crashed *LMirror*. In this situation the *LMaster* selects the *Worker* that is to become the next *LMirror*. For this it utilizes information about *Worker* resources, as well as contract and trust information (selected agent should not only be one of the best computers in the team, but have a contract with no expiration date and a very good performance track record). Next, the task that this *Worker* was working on is transferred to another *Worker*, while the selected *Worker* is upgraded by uploading appropriate modules (see, [18]. Among these modules the new *LMirror* obtains information where the team data is stored and details how to access this information. As soon as the new *LMirror* is fully operational, the *LMaster* undertakes the following steps: (a) informs the *CIC* about the fact that its team has the new *LMirror*, (b) informs all remaining *Workers* about the identity of the new *LMirror*, and (c) if necessary, informs the data storage facility that the new *LMirror* has access rights to the team data.

As noted above, the replacement of the crashed *LMaster* consists of two phases. First phase proceeds exactly as described in [28], and takes place within the node that hosts the *LMirror*. The only differences involve (i) access rights to the team information being established / confirmed with the data warehousing infrastructure, and (ii) copying necessary managerial data to the new *LMaster* (see, Section 4.6). The result of this phase is replacement of the *LMirror* (which at the end of the process self-destructs) by th new *LMaster*. This means that the team has now an *LMaster* and no *LMirror*. Therefore the, above described, procedure of re-creating an *LMirror* follows immediately, in the second phase.

**5. Concluding remarks.** The aim of this paper was to discuss issues involved in information mirroring in an agent-based Grid resource management system. We have focused our attention on information generated within the agent team and considered four important cases: (1) team data, (2) job contracts and their execution, (3) trust-related information, and (4) other sources of large volume information. We have established that we

have to deal with two main situations: (a) small-volume data that has to be mirrored immediately, and (b) large volume data that may be mirrored infrequently. Further analysis indicated that large volume data collection may be best achieved through utilization of a contracted data storage facility. This latter solution is our solution of choice and we plan to utilize it in our system.

**Acknowledgments.** Work of Maria Ganzha and Michal Drozdowicz was supported from the "Funds for Science" of the Polish Ministry for Science and Higher Education for years 2008-2011, as a research project (contract number N516 382434). Collaboration of the Polish and Bulgarian teams is partially supported by the *Parallel and Distributed Computing Practices* grant. Collaboration of Polish and French teams is partially supported by the PICS grant *New Methods for Balancing Loads and Scheduling Jobs in the Grid and Dedicated Systems*. Collaboration of the Polish and Russian teams is partially supported by the *Efficient use of Computational Grids* grant. Work of Marcin Paprzycki and Sofiya Ivanovska was supported in part by the National Science Fund of Bulgaria under Grant No. D002-146/16.12.2008

REFERENCES

[1] *Fast-start failover best practices: Oracle data guard 10g release 2*. http://www.oracle.com/technology/deploy/availability/pdf/MAA_WP_10gR2_FastStartFailoverBestPractices.pdf.
[2] *Paypal*. http://www.paypal.com.
[3] *Project: Minimum intrusion grid*. http://www.migrid.org/MiG/Mig/published_papers.html.
[4] *Sql server 2008 failover clustering*. http://download.microsoft.com/download/6/9/D/69D1FEA7-5B42-437A-B3BA-A4AD13E34EF6/SQLServer2008FailoverCluster.docx.
[5] *Welcome to the FIPA*. http://www.fipa.org/.
[6] A. Abraham, R. Buyya, and B. Nath, *Natures heuristics for scheduling jobs on computational grids*, in Proc. of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), 2000, pp. 45–52.
[7] H.-J. Burckert, K. Fischer, and G. Vierke, *Holonic transport scheduling with teletruck*, Applied Artificial Intelligence, 14 (2000), pp. 697–725.
[8] S. Bussmann and K. Schild, *An agent-based approach to the control of flexible production systems*, in Proc. of the 8th IEEE Int. Conf. on Emergent Technologies and Factory Automation (ETFA 2001), vol. 2, IEEE CS Press, Los Alamitos, CA, 2001, pp. 481–488.
[9] 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, 14 (2002), pp. 1507–1542.
[10] R. Buyya, J. Giddy, and D. Abramson, *An evaluation of economy-based resource trading and scheduling on computational power grids for parameter weep applications*, in Proceedings of the Second Workshop on Active Middleware Services (AMS 2000), Pittsburgh, USA, August 2000, Kluwer Academic Press.
[11] R. Cervenka and I. Trencansky, *Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS*, Whitestein Series in Software Agent Technologies and Autonomic Computing, A Birkhauser book, 2007.
[12] M. Dominiak, M. Ganzha, M. Gawinecki, W. Kuranowski, M. Paprzycki, S. Margenov, and I. Lirkov, *Utilizing agent teams in grid resource brokering*, International Transactions on Systems Science and Applications, 3 (2008), pp. 296–306.
[13] M. Dominiak, M. Ganzha, and M. Paprzycki, *Selecting grid-agent-team to execute user-job—initial solution*, in Proc. of the Conference on Complex, Intelligent and Software Intensive Systems, Los Alamitos, CA, 2007, IEEE CS Press, pp. 249–256.
[14] M. Dominiak, W. Kuranowski, M. Gawinecki, M. Ganzha, and M. Paprzycki, *Utilizing agent teams in grid resource management—preliminary considerations*, in Proc. of the IEEE J. V. Atanasoff Conference, Los Alamitos, CA, 2006, IEEE CS Press, pp. 46–51.
[15] M. Drozdowicz, M. Ganzha, W. Kuranowski, M. Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, M. Senobari, and I. Lirkov, *Software agents in adaj: Load balancing in a distributed environment*, Applications of Mathematics in Engineering and Economics'34, (2008), pp. 527–540.
[16] M. Drozdowicz, M. Ganzha, M. Paprzycki, M. Gawinecki, and A. Legalov, *Information flow and usage in an e-shop operating within an agent-based e-commerce system*, in Journal of Siberian Federal Univercity, vol. 2 of Engineering and Technologies, 2009, pp. 3–22.
[17] M. Drozdowicz, M. Ganzha, M. Paprzycki, R. Olejnik, I. Lirkov, P. Telegin, and M. Senobari, *Ontologies, agents and the grid—an overview*, in Profeedings of the PARENG'2009 Conference, 2009. in press.
[18] M. Ganzha, M. Gawinecki, M. Szymczak, G. Frackowiak, M. Paprzycki, M.-W. Park, Y.-S. Han, and Y. Sohn, *Generic framework for agent adaptability and utilization in a virtual organization—preliminary considerations*, in Proceedings of the 2008 WEBIST Conference, J. et. al., ed., Setubal, Portugal, 2008, INSTICC Press.
[19] M. Ganzha, M. Paprzycki, M. Drozdowicz, M. Senobari, I. Lirkov, S. Ivanovska, R. Olejnik, and P. Telegin, *Information flow and mirroring in an agent-based grid resource brokering system*, in Proceedings of LLSC Meetings. to appear.
[20] M. Ganzha, M. Paprzycki, and I. Lirkov, *Trust management in an agent-based grid resource brokering system—preliminary considerations*, in Applications of Mathematics in Engineering and Economics'33, M. Todorov, ed., vol. 946 of AIP Conf. Proc., College Park, MD, 2007, American Institute of Physics, pp. 35–46.
[21] N. Garg, D. Grosu, and V. Chaudhary, *Antisocial behavior of agents in scheduling mechanisms*, IEEE Transactions on Systems, Man and Cybernetics, 37 (2007), pp. 946–954. Part A.

[22] M. Ji, A. Veitch, and J. Wilkes, *Seneca: remote mirroring done write*, in HP Labs publications, 2003.
[23] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, *Designing of disaster*, in HP Labs publications, 2004.
[24] R. Kimball and M. Ross, *The data warehouse toolkit: the complete gide to dimensional modeling*, Wiley, 2 ed., 2002.
[25] W. Kuranowski, M. Ganzha, M. Gawinecki, M. Paprzycki, I. Lirkov, and S. Margenov, *Forming and managing agent teams acting as resource brokers in the grid—preliminary considerations*, International Journal of Computational Intelligence Research, 4 (2008), pp. 9–16.
[26] W. Kuranowski, M. Ganzha, M. Paprzycki, and I. Lirkov, *Supervising agent team an agent-based grid resource brokering system—initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., IEEE CS Press, Los Alamitos, CA, pp. 321–326.
[27] ———, *Supervising agent team an agent-based grid resource brokering system—initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., Los Alamitos, CA, 2008, IEEE CS Press, pp. 321–326.
[28] ———, *Supervising agent team an agent-based grid resource brokering system-initial solution*, in Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems, F. Xhafa and L. Barolli, eds., Los Alamitos, CA, 2008, IEEE CS Press, pp. 321–326.
[29] 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 LSSC Meeting, LNCS, Springer, 2007.
[30] M.Drozdowicz, M.Ganzha, M.Paprzycki, R.Olejnik, I.Lirkov, P.Telegin, and M.Senobari, *Parallel, Distributed and Grid Computing for Engineering*, Computational Scientce, Engineering and Technology Series:21, Saxe-Coburg Publications, Stirligshire, UK, 2009, ch. Ontologies, Agents and the Grid: An Overview, pp. 117–140.
[31] M.Senobari, M.Drozdowicz, M.Ganzha, M.Paprzycki, R.Olejnik, I.Lirkov, P.Telegin, and N.M.Charkari, *Parallel, Distributed and Grid Computing for Engineering*, Computational Scientce, Engineering and Technology Series:21, Saxe-Coburg Publications, Stirligshire, UK, 2009, ch. Resource Management in Grids: Overview and a discussion of a possible approach for a Agent-Based Middleware, pp. 141–164.
[32] 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, vol. 3470/2005 of Lecture Notes in Computer Science, Germany, 2005, Springer Verlag, pp. 651–660.
[33] R. H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara, *Snapmirror: File-system-based asynchronous mirroring for disaster recovery*, in FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies, Berkeley, CA, USA, 2002, USENIX Association, p. 9.
[34] D. A. Rajkumar Buyya and S. Venugopal, *The grid economy*, in Proceedings of the IEEE, vol. 93, 2005, pp. 698–714.
[35] M. Senobari, M. Drozdowicz, M. Paprzycki, W. Kuranowski, M. Ganzha, R. Olejnik, and I. Lirkov, *Combining an jade-agent-based grid infrastructure with the globus middleware—initial solution*, in Proc. of the CIMCA-IAWITC 2008 Conference, M. Mohammadian, ed., Los Alamitos, CA, 2008, IEEE CS Press, pp. 890–895.
[36] D. Ward and H. Hexmoor, *Deception as a means for power among collaborative agents*, in Proceedings of the Fifth International Symposium on Collaborative Technologies and Systems (CTS 2004), W. Smari and W. McQuay, eds., Society for Modeling and Simulation International, 2004, pp. 109–115.
[37] W. Xing, M. D. Dikaiakos, R. Sakellariou, S. Orlando, and D. Laforenza, *Design and development of a core grid ontology*, in Proc. of the CoreGRID Workshop "Integrated research in Grid Computing,", November 2005, pp. 21–31.

*Edited by:* Dana Petcu
*Received:* Oct 15th, 2009
*Accepted:* Nov 3rd, 2009