Combining AiG Agents with Unicore grid for improvement of user support

Kamil Łysik
Warsaw University Of Technology
Department of Mathematics and Information Sciences, Warsaw, Poland

Katarzyna Wasielewska, Marcin Paprzycki, Michał Drozdowicz
Systems Research Institute Polish Academy of Sciences,
Warsaw, Poland
Email: marcin.paprzycki@ibspan.waw.pl

Maria Ganzha
Institute of Informatics, University of Gdansk
Gdansk, Poland
System Research Institute Polish Academy of Sciences, Warsaw, Poland
Email: mganzha@inf.ug.edu.pl

John Brennan, Violetta Holmes, Ibad Kureshi
University of Huddersfield, Huddersfield, UK
Email: V.Holmes@hud.ac.uk

***Abstract*—Grid computing has, in recent history, become an invaluable tool for scientific research. As grid middleware has matured, considerations have extended beyond the core functionality, towards greater usability. The aim of this paper is to consider how resources that are available to the users across the Queensgate Grid (QGG) at the University of Huddersfield (UoH), could be accessed with the help of an ontology-driven interface. The interface is a part of the *Agent in Grid* (*AiG*) project under development at the Systems Research Institute Polish Academy of Sciences (SRIPAS). It is to be customized and integrated with the UoH computing environment. The overarching goal is to help users of the grid infrastructure. The secondary goals are: (i) to improve the performance of the system, and (ii) to equalize the distribution of work among resources. Results presented in this paper include the new ontology that is being developed for the grid at the UoH, and the description of issues encountered during the development of a scenario when user searches for an appropriate resource within the Unicore grid middleware and submits job to be executed on such resource.**

I. Introduction

A. Background

In many educational and scientific institutions, the grid middleware of choice is the Globus Toolkit (GT; [1]), which is used to unify disparate, or geographically displaced, systems. While the Globus is a mature and efficient grid middleware that is often applied for linking computing systems, it lacks key utilities e.g. resource discovery for users, which makes it a relatively user-unfriendly system. As a result, burden is placed on system administrators to continually publish up to date resource information so that users are able to submit jobs to the systems. Another aspect of this problem is that the GT is unaware of the underlying systems and, for example, it will keep submitting jobs to a cluster even if there are problems with that resource.

Automated resource allocation was of real interest to the administrators of the University of Huddersfield's (UoH) Queensgate Grid (QGG). This grid consists of a number of distinct resources both on-campus and through resource sharing agreements with regional consortia [2], [3]. Note that, most of these systems utilize different job schedulers and this creates a steep learning curve for the average user. Using the standard GT, users can submit their jobs using a single scripting language. However, first they need to be aware of the existing system(s) (to be able to connect to it(them)). This arrangement does not allow for the best use of resources, as users will often submit jobs to the systems they are "used to," possibly overlooking better alternatives.

The European Middleware Initiative (EMI; [4]) middleware holds a solution to this situation by enabling the job submission system to communicate directly with the information gathering systems, allowing up to date resource information to be collected, including current system loads, thus allowing for much more intelligent decisions to be made, as to which resource will be best suited for a given job. This automated resource discovery allows users to specify required computing attributes, during the submission and, based upon these requirements, the EMI system will automatically select the most appropriate resource within the grid to complete the needed processes. The system can then translate the job specification into the required job description language for the batch system, whether that is PBS, LSF or SGE, and directly submit to the endpoint [5]. In theory, the EMI middleware enables better use of available resources by removing the burden of allocation from the user [6], while using dynamic information concerning current system loads.

The design of the EMI system implemented at the UoH was based around five local servers and two national level services provided by the National e-Infrastructure Service (NES) [7], [8]. The local servers comprised of a User Interface (UI), Workload Management System (WMS), Site level Berkeley Database Information Index (BDII), Top level BDII, and a MyProxy server. The system makes use of the NES based Virtual Organisation Management System (VOMS) server,

978-1-4799-2795-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CANDAR.2013.18

TABLE I
THREE SYSTEM USAGE IN THE LAST 6 MONTHS OF 2012.

Month	Eridani	Sol	Condor	Jobs
7	1	0	0	1
8	41	28	0	69
9	42	3068	0	3110
10	4216	11017	200	15433
11	3686	5831	5612	15129
12	447	1440	2050	3937
Total	8433	21384	7862	37679

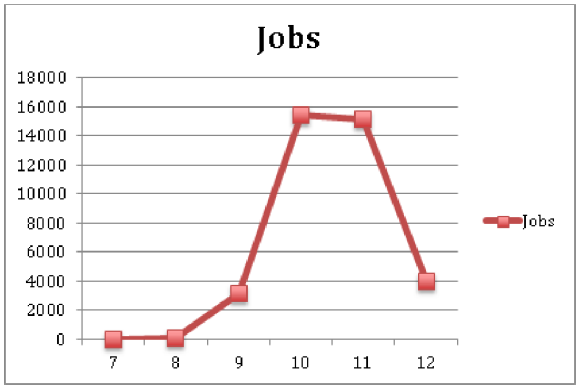

Fig. 1. Job Volumes per Month.

responsible for giving Virtual Organization (VO) attributes to user credentials. Currently all user credentials are supplied through the NES Certificate Authority (CA), although there are plans to create local implementations of these services to allow for greater control over local users.

However, as it will be argued in Section I-B, even installation of the EMI middleware did not result in the expected improvement of effectiveness of grid use. Therefore, a different approach to facilitating job submission is being considered. In this paper we summarize initial considerations concerning use of ontologies and semantic data processing to facilitate user support in the UoH grid. Applied semantic technologies originate from the *Agents in Grid* (*AiG*) project pursued by the researchers of the Systems Research Institute Polish Academy of Sciences (SRIPAS). In this paper we discuss how the *AiG* infrastructure can be combined with the Unicore grid moddleware, which is one of the middlewares used within the UoH grid infrastructure.

B. Motivation

The context for this paper is provided by the issues encountered at the Univerisity of Huddersfield. Researchers at the UoH have access to multiple systems on campus including clusters, a Condor pool, Windows and Linux storage resources, along with many resources available off campus such as the enCore [9], [10] (a collaboration between the STFC, the University of Huddersfield and the OCF), along with resources hosted by the NGS and members of the North West Grid (NW-Grid) [11]. With so many systems available in the grid, in the case when there is *no grid wide resource discovery* it can be difficult to effectively use and manage these systems. Observe that in this situation, grid users can only connect/submit jobs to the systems that they are aware of (e.g. they heard off during a training that took place some time in the past). Therefore, a heavy burden is placed upon the administrators to supply the information about available systems, and developing a model for systematic delivery of updated information. Even with this in place, in order to complete their research, users tend to stay with one system/submission method that they have learned in the past, unless something goes wrong with that method. This can lead to lengthy queues on some systems, or even exceeding contractual monthly hours at some remote sites, while other systems (available in the grid) stay underutilized. This fact can be illustrated in the following data, collected at the UoH. Consider tables Table I and Fig. 1. There it can be seen that the number of jobs on the QGG hardly changed through the October and November of 2012. However, looking at the data in Fig. 2 it can be seen that the job distribution in November shifted towards a much more balanced one. The only, known to us, reason for this shift was that in November the QGG administrators personally informed users that some of their software was available on the HTCondor system and supplied users with sample job scripts. As a result some users decided to migrate their work (primarily from the *Sol* node) to the Condor pool.

This being the case, a decision was made to create a "single view" system that would link all available, geographically distributed, HPC resources. Currently many open source grid middleware solutions are available; such as EMI, Unicore, or Globus Toolkit, among others. For instance, Unicore is a mature grid middleware which, like the EMI, can interoperate with other middlewares allowing it to be placed above the GT in a software stack, or be deployed as a complete ("stand-alone") solution. However, these middlewares still lack the "single point of contact" offered by the EMI and the job submission needs to be targeted to specific systems, instead of having a WMS-type service to do that. Therefore, the EMI was chosen as the middleware that can be implemented on top of the current software stack with minor modifications to the Compute Element(s) (CE). As a result, installation of the EMI, would have no impact on the standard submission procedures, thus allowing the new system to be brought online and tested without impacting ongoing workflows and also be compatible with the existing NGS and the European e-infrastructure. As a result, the decision was made to implement the EMI on the QGG and, in this context, a base configuration needed to be determined. Through the research carried out for the project it was found that five local servers would be sufficient to produce a working system. Fig. 3 shows the required base configuration to setup an EMI based grid to serve all functions, apart from those that were to be handled by the external VOMS server and the CA.

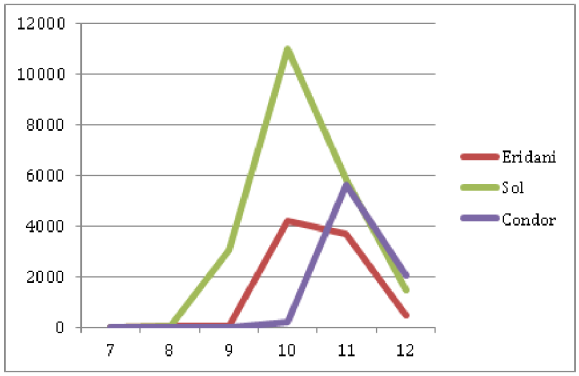

Fig. 2. Job Distribution Across Three Systems.

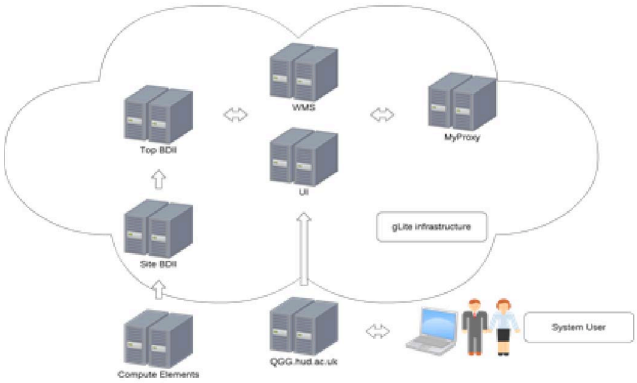

Fig. 3. EMI Deployment within the QGG.

The aims of this project were to unify the resources available to the users of the QGG, while simultaneously reducing the, above mentioned, burdens placed on the administrators, for publishing up to date resource information and training users how to use multiple submission systems. Furthermore, it was considered to be highly beneficial for the system to integrate it as seamlessly as possible with the national level infrastructure, to simplify resource sharing for any future collaborative work. Ideally users would need only a single point of access to the QGG, from where the information about the current state of all available resources would be available, and could be matched against job requirements, with users only being aware of a single job description language. In this way, using of the EMI should also simplify the training requirements for staff, students and post graduate researchers, as there would be only a single job description language to teach the vast majority of users.

However, with time it was realized that even this approach would not be enough to satisfy the real-life user requirements. As stated above, experiences gathered at the UoH, and prevalent across all computing centers that the authors of this paper contacted, are as follows. Users can be divided into two categories: (1) small group of "active users" who are ready and willing to explore possibilities and expand their "toolbox" of known methods of completing their computational jobs, and (2) majority of users who are interested only in solving their problems; are not interested in spending time and energy to learn new things related to available computing infrastructures; may have minimal knowledge of computer programming / use (e.g. art professors who use 3D rendering programs). Taking this into account, there arose a need to develop an even more user-friendly and easy to manage interface that would support users in finding a resource with specified configuration and help them to submit a job (without knowledge of *any* job description language). Furthermore, job submission process should be the same regardless of the grid middleware installed on a given resource. To achieve this goal it was proposed to integrate the *Agents in Grid* (*AiG* [12]–[14]) middleware with the EMI based grid available at the UoH, to provide dynamic ontology-driven user interface (see, [15] for more details). In the remainder of this paper we discuss how it should be possible to combine the *AiG* semantic-agent system with the UoH available grid middleware. To provide the context of the discussion, while simplifying somewhat the situation, we assume that all resources have Unicore middleware that is also accessible from within the EMI.

C. Agents in Grid project

Let us start from a brief description of the basic features of the *Agents in Grid* system, which aims at the development of an agent-based infrastructure for intelligent resource management in the grid. One of the main assumptions of the *AiG* is to use ontologies and semantic data processing to represent all knowledge in the system, as well as the content of messages exchanged by system components. Therefore, we have developed an ontology of the grid combined with ontologies needed for contract negotiations (for details, see [16]). This ontology modifies and extends the ontology developed during the CoreGrid project. Originally, in the *AiG* system has been developed for the open grid (e.g. the Internet). Therefore, it was suggested that, to overcome certain possible problems), agents work in teams. This resulted in the need to consider two main scenarios: (i) user wants to execute a job using a resource available in the grid (and this resource becomes available through one of the existing team), (ii) worker wants to join a team to make its resource(s) available. Overall, system is based on the following tenets (in what follows, we will discuss changes needed to adapt the *AiG* system to work with grid middleware in a controlled environment of the UoH):

- users are represented by their "personal agents" (*LAgents*),
- agents work in teams (groups of agents representing resources) and in this way they avoid problems caused by resource disappearance,
- each team has a single leader – *LMaster agent*,
- each *LMaster* has a mirror *LMirror agent* that can take over its job in the case when the *LMaster agent* crashes,
- incoming workers (*worker agents* – instances of *LAgents*) – join teams based on user-defined criteria,

- teams (represented by *LMaster*s) accept workers based on criteria internal to the team,
- each *LAgent* can (if needed) play role of the *LMirror*, or of the *LMaster*,
- matchmaking (between the agent seeking resources to execute a job, or wanting to sell its resources and teams existing in the system that can execute a job or seek workers) is facilitated by the *CIC* component (in the system represented by the *CIC agent*),
- terms of collaboration (*Service Level Agreement*) between the *LAgent* (trying to execute a job, or representing the resource) and a team represented by *LMaster* result from negotiations.

Further discussion concerning the rationale behind the above listed assumptions, and elaboration of their practical realization, can be found in [12]–[14], [16] and papers referenced there.

It is easy to observe that the full functionality provided by the *AiG* system is much more extensive than what is needed to provide ontology-based user support in the scenario / environment described above. This is caused, among others, by the fact that the *AiG* system is prepared to work in an uncontrolled "open grid" environment, while at the UoH we deal with closed grid, controlled by appointed administrators. Therefore, the *AiG* has to be customized to support a scenario shown in Fig. 4. Here, let us note that, instead of teams

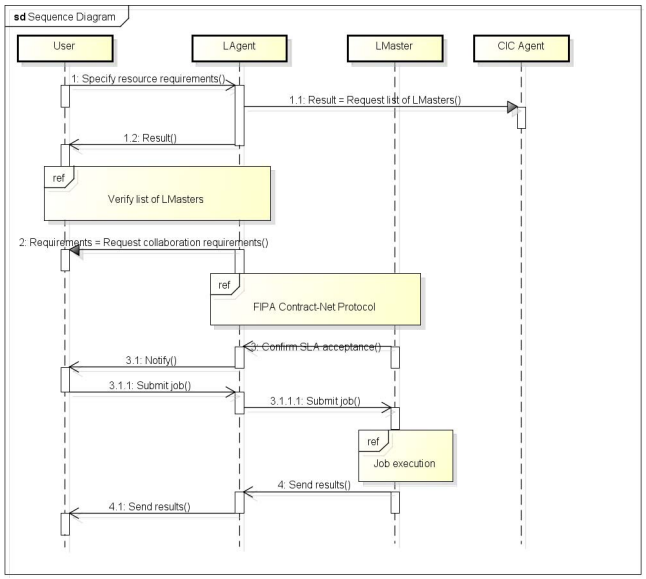

Fig. 4. Sequence diagram with a simplified job execution scenario from the *AiG* system.

represented by their *LMaster*s, we could have (among others):

1) an *AiG* interface communicating with a single grid middleware (e.g. the Unicore – the example here) represented by a single *grid agent* – counterpart to the *LMaster* – here, the primary reason for the use of the *AiG* software would be to apply the ontology-driven front-end;
2) an *AiG* interface communicating with a multiple grid middlewares, each represented by a single *grid agent* – counterpart to the *LMaster* – here, in addition to the ontology-driven front-end the system could negotiate which grid middleware would have the needed resource and be available at the time-frame requested by the user.

Here, further discussion is in order. In what follows we discuss how the *AiG* middleware can be combined with the Unicore. However, in Section I-B we have specified that the ultimate goal for the UoH would be to provide an ontology-driven front-end for the EMI middleware. There is a number of reasons for proceedings the way we do. First, the EMI middleware is quite complex, as it is actually a meta-level grid middleware that "contains" the Unicore as one of its components. Therefore, our current work is a stepping-stone toward the full EMI integration. Second, there exist sites that, for a variety of reasons, to not plan to deploy the EMI (e.g. while running the Unicore). This being the case, the AiG infrastructure should be able to interact with any actual grid middleware, and this paper illustrates that this is feasible. Finally, it is possible to use only the AiG ontological front-end (see, below) and interface it with the EMI infrastructure (and this may be the ultimate solution for the UoH). However, in this case only an EMI-specific solution would be tried, without possibility of providing a single point of contact infrastructure, consisting of any combination of grid middlewares and (possibly) not-gridified resources. Therefore, material presented in this paper should be seen as an exploratory work towards, on the one had, a solution to be installed at the UoH, and a general approach for fusion of *AiG* components with any existing grid middleware, on the other.

Returning back to the main line of reasoning, we note that since there are no agent teams, the team joining functionality can be eliminated from further considerations. Furthermore, negotiation of terms of collaboration between the *LAgent* representing the user and the *grid agent*s can be simplified to verifying: (i) if there is(are) a needed resource(s) in the given grid; and (ii) if the job can be executed within given time frame (here, for instance, the *LAgent* could select the grid that would execute its job earlier). Therefore, there will be no need to negotiate economic trends of job execution: e.g. price, deadline penalty etc. Use case diagram of the simplified system in shown in Fig. 5.

II. Integration of AiG with Unicore grid

As proposed above, for integrating the *AiG* infrastructure with the Unicore grid a special *grid agent* was created. Its purpose is to serve as a proxy between the Unicore grid and the *AiG* system and to manage / oversee the state of the job execution. Obviously, the implementation of the *grid agent* depends on the supported grid middleware. Each system requires its own *grid agent* implementation. However, thanks to the agent-based approach, it is possible to easily extend the *AiG* to support multiple grid middlewares, without changing the implementation of the other agents. The only addition to

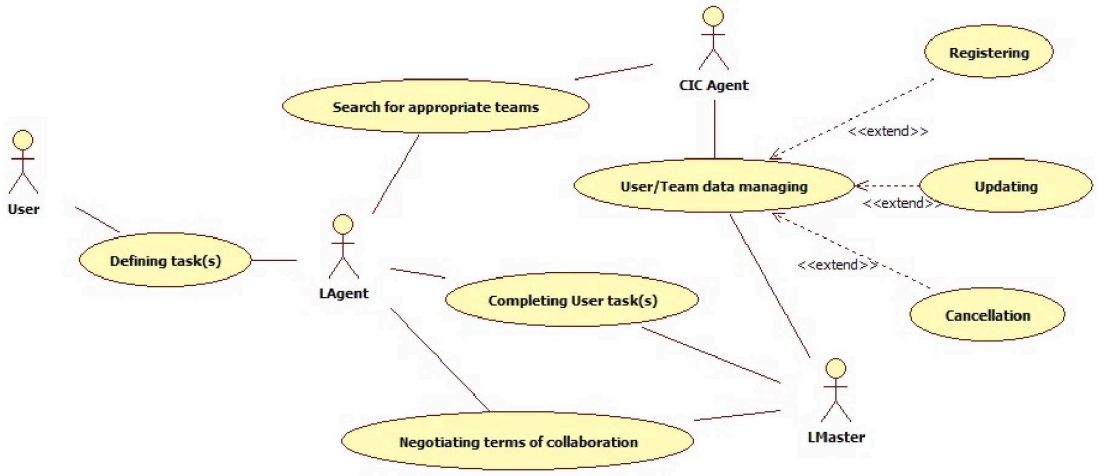

Fig. 5. Simplified use case diagram of *AiG* system.

the *AiG* system would be the *grid agent* corresponding to a given (new) grid. Thus, the integration of the *AiG* with the EMI grid would require creation of a single *EMI grid agent*. Here, it should be observed that this method of initial integration of the *AiG* with the EMI may be simpler that fusing the *AiG* ontological front-end with it. This is because the agent-level integration could proceed without the need to directly interact with the EMI code (only meta-level information would be exchanged, in a similar way that in the case of the *Unicore grid agent* described below). However, this issue requires further investigation.

Here, note that the scalability of the proposed approach, while not experimentally tested, should not be a big problem. First, in [17], [18] it was shown that the JADE agents (which are used in the *AiG* system) are highly scalable. Second, in the newest version of JADE agent platform, a new mechanism of "agent aliases" was added. It allows to instantiate multiple agents that are known to the system under the same name. In this way, if needed, multiple copies of the *grid agent* can be instantiated to handle the workload.

Concluding this section let us observe that the scenario that has to be implemented is as follows. User specifies the job execution requirements (interacting with the ontology-driven front-end; described in the next two sections). These requirements are received (from the interface) by the *LAgent* (representing the user). Depending on the situation (see above), the *LAgent* may or may not need to negotiate, which grid is going to be used to execute the job. Next, it forwards the job information to the appropriate *grid agent*. The job submission contains also all required parameters and input data. At this stage, the job "leaves" the *AiG* system, as it is submitted (by the *grid agent*) to "its" grid middleware. However, the *grid agent* monitors the state of the job execution (using mechanisms available in the grid). If needed, the *grid agent* can send information about the job state to the appropriate *LAgent*. When the job execution is finished the *LAgent* agent is informed about the results. If the job failed, the *LAgent* is informed about the failure (with the error message included). In case of successful execution, the *grid agent* retrieves the results from the grid and sends them to the *LAgent*. Let us now look into this process in more detail.

A. Implementing Ontologies

The first issue that we have to address is the question of ontologies used in the system. As stated above, in the *AiG* project we have modified and considerably extended the Core Grid Ontology. This was to be able to apply semantic data processing across the system. When considering integration of the *AiG* system with any other software artifact we have to make sure that the ontologies used in the system will be able to properly capture "features" of the new software. Therefore, one of the most crucial steps in integrating the *AiG* agents with the UoH QGG grid is to implement dedicated ontologies. In the *AiG* system, three main ontologies are used: (i) *AiG Grid Ontology* – to describe resources and structure of the grid, (ii) *AiG Conditions Ontology* – to describe contracts (terms of collaboration) between grid users and resource providers, (iii) *AiG Messaging Ontology* – to describe content of messages exchanged in the system. These ontologies provide the conceptual model that is later used to prepare the deployment-specific ontologies with instances and assertions about them (for more information about the *AiG* ontologies, see [16]).

¿From the ontological perspective, the integration requires preparation of a new (dedicated) ontology with instances describing the QGG grid structure, verification that the Unicore grid specific parameters exist within the *AiG Grid Ontology*, and inclusion of application specific parameters to describe the job before its submission. Here, to test the *AiG* in the rendering job (which is one of typical applications ran in the UoH grid) we used the POV-RAY raytracing engine (instead of the Mental Ray that is actually used in the UoH). The POV-RAY

requires that the input file contains the input data. Additionally, rendering parameters can be passed as application options, like output image size, or output file format. For instance, the POV-RAY execution command could have the form: `povray image.pov +W1024 +H1024`, where +W is the width, and +H is the height of the rendered image. Since the POV-RAY was used only in our initial prototype, we will not list the application-specific ontology, as it will have to be adjusted when, in the next step of the project (we will create and use the "Mental Ray ontology").

To illustrate how the UoH grid-available hardware was modeled in the dedicated UoH ontology, the following code snippet shows a fragment of the ontolgical description of the UoH grid infrastructure. Here, we see the individuals that represent the grid middleware (Globus 4, Globus 5 and Unicore 6) and computing elements (equivalent to grid systems) actually available in the UoH grid, named TAUCETI, CONDOR POOL and UNICORE. For each computing element, a common configuration can be specified e.g. Meta scheduling service for the UNICORE. Moreover, each computing element has a set of worker nodes (units within the given grid system represented with dedicated *LAgent*s that can execute a job) e.g. the UNICORE ComputeNode 1 is a worker node within the UNICORE grid system that has the indicated CPU, memory and storage space. Finally, we present only a part of the configuration of the selected computing elements and worker nodes.

```
<owl:NamedIndividual rdf:about="&UoHGridInstances;
    Globus4">
 <rdf:type rdf:resource="&AiGGridOntology;Globus4"/>
 <cgo:hasComponent rdf:resource="&UoHGridInstances;
     CondorPool"/>
 <AiGGridOntology:hasGridService rdf:resource="&
     UoHGridInstances;Scheduler_Troque_Maui"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    Globus5">
 <rdf:type rdf:resource="&AiGGridOntology;Globus5"/>
 <cgo:hasComponent rdf:resource="&UoHGridInstances;
     TAUCETI"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    Unicore6">
 <rdf:type rdf:resource="&AiGGridOntology;Unicore6"/
     >
 <cgo:hasComponent rdf:resource="&UoHGridInstances;
     UNICORE"/>
 <AiGGridOntology:hasGridService rdf:resource="&
     UoHGridInstances;MetaScheduling_Service"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    TAUCETI">
 <rdf:type rdf:resource="&cgo;ComputingElement"/>
 <rdfs:label xml:lang="en">TAUCETI</rdfs:label>
 <AiGGridOntology:hasOperatingSystem rdf:resource="&
     UoHGridInstances;CentosOS6"/>
 <AiGGridOntology:hasGridService rdf:resource="&
     UoHGridInstances;Scheduler_Troque_Maui"/>
 <cgo:hasWN rdf:resource="&UoHGridInstances;
     TAUCETI_ComputeNode_1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    TAUCETI_ComputeNode_1">
 <rdf:type rdf:resource="&cgo;WorkerNode"/>
 <rdfs:label xml:lang="en">TAUCETI Compute Node #1</
     rdfs:label>
 <cgo:hasCPU rdf:resource="&UoHGridInstances;
     CPU_2DualCoreAMDOpteron275"/>
 <AiGGridOntology:hasOperatingSystem rdf:resource="&
     UoHGridInstances;CentosOS6"/>
 <AiGGridOntology:hasMemory rdf:resource="&
     UoHGridInstances;Memory_3GB_RAM"/>
 <AiGGridOntology:hasStorageSpace rdf:resource="&
     UoHGridInstances;Storage_65GB_HDD"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    CondorPool">
 <rdf:type rdf:resource="&cgo;ComputingElement"/>
 <rdfs:label xml:lang="en">CONDOR POOL</rdfs:label>
 <AiGGridOntology:hasGridService rdf:resource="&
     UoHGridInstances;Condor_7.6.7"/>
 <AiGGridOntology:hasOperatingSystem rdf:resource="&
     UoHGridInstances;Windows"/>
 <cgo:hasWN rdf:resource="&UoHGridInstances;
     CONDOR_POOL_ComputeNode_1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    CONDOR_POOL_ComputeNode_1">
 <rdf:type rdf:resource="&cgo;WorkerNode"/>
 <rdfs:label xml:lang="en">CONDOR POOL Compute Node
     #1</rdfs:label>
 <cgo:hasCPU rdf:resource="&UoHGridInstances;
     CPU_IntelCorei52320"/>
 <AiGGridOntology:hasMemory rdf:resource="&
     UoHGridInstances;Memory_4GB_RAM"/>
 <AiGGridOntology:hasStorageSpace rdf:resource="&
     UoHGridInstances;Storage_500GB_SATA3"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    UNICORE">
 <rdf:type rdf:resource="&cgo;ComputingElement"/>
 <rdfs:label xml:lang="en">UNICORE</rdfs:label>
 <AiGGridOntology:hasOperatingSystem rdf:resource="&
     UoHGridInstances;ScientificLinux"/>
 <AiGGridOntology:hasGridService rdf:resource="&
     UoHGridInstances;MetaScheduling_Service"/>
 <cgo:hasWN rdf:resource="&UoHGridInstances;
     UNICORE_ComputeNode_1"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    UNICORE_ComputeNode_1">
 <rdf:type rdf:resource="&cgo;WorkerNode"/>
 <rdfs:label xml:lang="en">UNICORE Compute Node #1</
     rdfs:label>
 <AiGGridOntology:hasOperatingSystem rdf:resource="&
     UoHGridInstances;ScientificLinux"/>
 <cgo:hasCPU rdf:resource="&UoHGridInstances;
     CPU_IntelCorei52320"/>
 <AiGGridOntology:hasMemory rdf:resource="&
     UoHGridInstances;Memory_1GB_RAM"/>
 <AiGGridOntology:hasStorageSpace rdf:resource="&
     UoHGridInstances;Storage_10GB_HDD"/>
</owl:NamedIndividual>

<owl:NamedIndividual rdf:about="&UoHGridInstances;
    Storage_500GB_SATA3">
 <rdf:type rdf:resource="&cgo;StorageSpace"/>
 <rdfs:label xml:lang="en">500 GB with SATA3 Disk</
     rdfs:label>
 <AiGGridOntology:hasAvailableSize rdf:datatype="&
     xsd;int">512000</
     AiGGridOntology:hasAvailableSize>
```

```
<AiGGridOntology:hasStorageInterface rdf:resource="
    &UoHGridInstances;SATA3"/>
</owl:NamedIndividual>
```

B. AiG ontology-driven user interface

One of the key features of the *AiG* system is the ontology-driven dynamic user interface that enables user without ontology knowledge to select a class or an individual on the basis of the ontology of the system. The challenge was to design a user-friendly mechanism to create OWL class expressions and individuals. This was achieved through the development of the *condition builder* that enables one to define conditions on multiple (different) properties of the root class. The user can select the property she wishes to restrict, choose an appropriate operator and type in, or select, the value for that property. The main advantage of such approach is that the system benefits directly from the semantic data processing and features of data representation within the OWL 2.0, while the semantic data processing is being hidden from the user. Furthermore, the flexible and adaptable interface allows for ontology modification without the need for modification of the underlying code of the system (front end) that utilizes it. Specifically, changes made to the ontology are immediately "loaded" and become visible to the user. This feature can greatly simplify work of grid administrators, who can make new hardware / software visible to the users by simply modifying the ontology. All interactions with the user actor in Fig. 4 utilize just described interface. Therefore, all input data specified by the user in "non-ontological interactions" with the interface, are translated into ontologically demarcated data that is processed within the system.

To illustrate the way that the interface operates, let us consider the first step of job execution. Here, the user specifies the requirements for the configuration of resources that shall be used to execute a job. After logging into the system, the user works with an interface similar the one shown in Fig. 6; to restrict the properties of the computing element. Note that, for the time being, this interface has not been developed with usability in mind. Therefore, Fig. 6 depicts what should be see as a rough representation (focused on the available data) of what can be shown to the user. Here, the underlying system

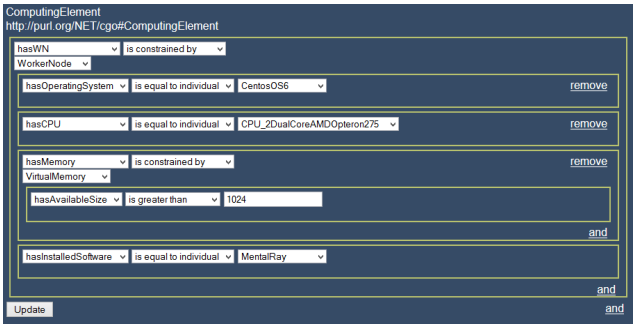

Fig. 6. Selection of resource configuration in the interface.

ontologies are the *AiG* conceptual ontologies and the dedicated UoH ontology, with the QGG grid structure and resource specification. Let us stress, that the presented information corresponds to the actual structure and node configuration of the UoH grid. Therefore, classes and properties are consistent with what would be specified for the Unicore grid configuration, and the user can specify configuration that should match one of the resources that are available (e.g. one of multiple computing nodes with the Unicore). Moreover, users can explicitly indicate the QGG computing node; if they know that it has the optimal configuration for their job (see, Fig. 7).

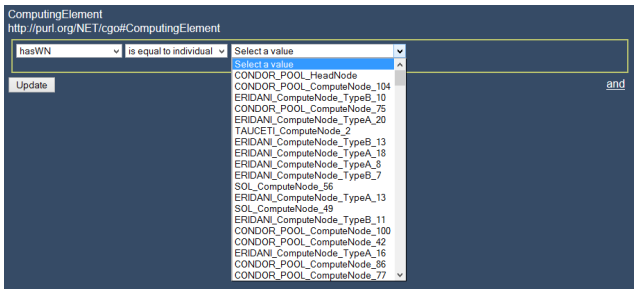

Fig. 7. Selection of a specific resource in the interface.

C. Interaction of JADE AiG agents with the Unicore grid middleware

Let us now consider the implementation of the *Unicore grid agent* that represents the Unicore grid in the *AiG* system. Specifically, the *Unicore grid agent* accepts job submissions provided by the *LAgent* (representing the user) and sends them to the Unicore grid. Next, it is monitoring the job execution status and taking care of the job execution results. Here, let us stress again, that the *Unicore grid agent* is an instance of a general concept of a *grid agent*. Specifically, to support a different grid middleware (e.g. to create a *gLite grid agent*) all that would be required is to create a new *grid agent* with a logic allowing a job to be submitted to the gLite grid; status of which should be monitored and results of which to be returned to the appropriate *AiG LAgent*. In this way, the *grid agent* is an interface to a selected grid middleware (it can be also regarded as a type of the *LMaster* agent with a limited / dedicated functionality).

The *Unicore grid agent* can communicate with other agents in the *AiG* system (i.e. *LAgent*s representing users) to receive job submissions. It is assumed that the *grid agent*, representing the grid middleware to execute the job, is selected during a negotiation process. Henceforth, *grid agent*s communicate also with "their own" grid installation.

In the current implementation of the *Unicore grid agent* only the Unicore 6 is supported. To execute the job the *Unicore grid agent* must receive a job submission document and a list of input files (uploaded to the grid). The job submission document is described in the *Job Submission Description Language* (JSDL) that is supported by many grid systems. The mandatory element of the submission document is the name of

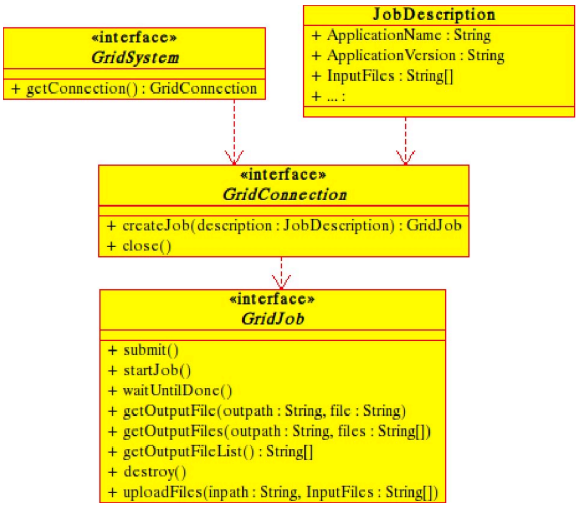

Fig. 8. AiG job execution interfaces

the application, i.e. the name of the software that is available in the grid and is requested to be executed (e.g. the POV-RAY). Without it the grid cannot determine what software should be used. Additionally, the JSDL provides many parameters to describe jobs, e.g. the command to execute, the resource requirements, or the job execution limits. As soon as the *grid agent* retrieves the required elements it can start arranging job execution.

Job execution is performed by the grid middleware represented by respective *grid agent*. When the job finishes, the *grid agent* knows about that fact and can access information about the execution status. In case of success, the *grid agent* accesses the job submission log and output files, and returns the results to the appropriate *LAgent*. Otherwise the *grid agent* reports to the *LAgent* that the job execution failed (and returns the error message). Here, note that, for the Unicore grid middleware, the returned results contain also the output of the standard streams (stdout, stderr) that are send as text strings. Obviously, it could be possible to return only information that the job has been completed and information about location of the output files. This approach, due to restrictions on file access and sharing, is rather inappropriate for the "open grid." However it will be considered for the UoH grid infrastructure, where the limiting factors are absent.

To make the *AiG* independent of the specific grid middleware, a new module was created. It contains all logic that is required to submit a job to the Unicore grid (and, in the future, other grid middlewares) and retrieve the results. It is based on the code shipped with the EMI grid middleware and specific parts of the Unicore system. Thanks to the EMI code, it was possible to reuse some Java classes for interaction with other grid systems / middlewares. Obviously, the main purpose for the creation of a separate module was to extract the grid execution logic as separate project and to integrate the execution services with the *AiG* project (or other systems, if necessary).

The new module provides methods to configure the *Unicore grid agent* and to submit job to the Unicore middleware.Yhe job submission code is as generic as possible, due to extraction of the class interfaces (see Fig. 8).

The first step of job execution is establishing connection to the configured grid, using the $getConnection()$ method. When the connection is acquired and the job description object is filled, the job can be submitted. First, the job is created to get the *GridJob* object. Next, to execute the job the following command is used:

```
GridJob job = connection.
    createJob(description);
job.submit();
job.uploadFiles(inputPath, inputFileNames);
job.startJob();
```

Here, the Unicore job submission mechanism creates a job in the grid, but does not start it. The file upload is possible only after the job is submitted and should be done before the job starts. To finish the job and to download the files, the following commands are issued:

```
job.waitUntilDone();
String[] outFileNames =
  job.getOutputFileList();
job.getOutputFiles(outputPath, outFileNames);
```

Note that the $outFileNames$ can be filtered here. For example, it is possible to get only the image files filtered from among all other file types (e.g. to omit the text files). By default, the library lists all files that are in the grid working directory.

Finally, the cleanup is required. For the Unicore grid middleware there will be a list of jobs that aren't deleted and will wait until the administrator deletes them manually. However, this should not happen if the *Unicore grid agent* has direct connection to the grid. Then it would take care of the problem by issuing the commands:

```
job.destroy();
connection.close();
```

Here, note that, to configure the connection to the Unicore grid middleware, it is required to set the connection URL and use credentials with certificates. To get the certificate and the URL it is necessary to contact the grid administrator. However, the *Unicore grid agent* configuration is performed only once for every grid installation and has to take place during each *Unicore grid agent* startup.

III. Concluding Remarks

The aim of the paper was to outline the motivation and actions undertaken to integrate the *AiG* agent-semantic system for management of grid resources, with the EMI grid middleware that is being installed and tried at the University of Huddersfield (UoH). We have started by outlining the reasons behind the need for application of an ontology driven interface to support users of the UoH grid resources. Next, we have presented the main features of the *AiG* system and discussed the way it has to be modified to match the requirements of the integration with the UoH grid. Here, we have considered different ways of integrating a semantic interface with the EMI

(and possibly other) grid middleware(s). Furthermore, we have used the Unicore grid middleware as the use case example to discuss how to combine the *AiG* system with any other grid middleware. While the integration has been completed and we have successfully run the POV-RAY application instantiated in a local grid at the SRIPAS, we have decided not to present this application as it does not bring any additional information other than that the integration was successful. In the next step we plan to fuse the *AiG* system with the EMI grid middleware used at the University of Huddersfield. Here, the above noticed research questions, e.g. if we should resign from the agents and use only the semantic interface module, or how to handle the results – to leave to be picked up, or to transfer to the *LAgent*, will be considered. Separately, integration of the *AiG* infrastructure with other popular grid middlewares (e.g. Globus Toolkit, gLite, CONDOR, etc.) will also be tried.

IV. Acknowledgement

The author(s) would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work. The authors would like to acknowledge the use of the UK National e-Infrastructure Service in carrying out this work and for the support provided by the NES helpdesk.

References

[1] Ian Foster, Carl Kesselman, Steven Tuecke, "The anatomy of the grid – enabling scalable virtual organizations," mar 2001. [Online]. Available: http://arxiv.org/abs/cs/0103025

[2] I. Kureshi, "Establishing a university grid for HPC applications – university of huddersfield repository," Thesis (Masters), University of Huddersfield, Huddersfield, 2010. [Online]. Available: http://eprints.hud.ac.uk/10169/

[3] V. Holmes and I. Kureshi, "Huddersfield university campus grid: QGG of OSCAR clusters," *Journal of Physics*, vol. 256, no. 1, 2010. [Online]. Available: http://iopscience.iop.org/1742-6596/256/1/012022/

[4] EMI Collaboration, "Emi, mna3.2 open source software initiative," jan 2013. [Online]. Available: http://cds.cern.ch/record/1450878/files/EMI-MNA3.2-1450878-OS_Initiative_Charter_v1.0.pdf?version=1

[5] Cristina Aiftimiei, Alberto Aimar, Andrea Ceccanti, Marco Cecchi, Alberto Di Meglio, Florida Estrella, Patrick Fuhrmann, Emidio Giorgio, Balazs Konya, Laurence Field, Jon Kerr Nilsen, Morris Riedel, and Morris Riedel, "Towards next generations of software for distributed infrastructures: the european middleware initiative." [Online]. Available: http://www.pd.infn.it/~aiftim/papers/emi.pdf

[6] S. Campana and D. Rebatto and A. Sciaba, "Experience with the gLite workload management system in ATLAS monte carlo production on LCG," *Journal of Physics*, vol. Volume 119, no. 5, 2008. [Online]. Available: http://iopscience.iop.org/1742-6596/119/5/052009/pdf/1742-6596_119_5_052009.pdf

[7] D. Wallom, P. Oliver, A. Richards, and S. Young, "NGS site level services," Sep. 2009. [Online]. Available: http://www.ngs.ac.uk/sites/default/files/NGS_Software_Stack-3-2.pdf

[8] J. Jensen, G. A. Stewart, M. Viljoen, D. Wallom, and S. Young, "Practical grid interoperability: GridPP and the national grid service," in *UK e-Science All Hands Conference*, 2007. [Online]. Available: http://install.gridpp.ac.uk/papers/GridInteoperability-AHM07.pdf

[9] J. Dixon, "enCORE on-demand HPC," http://www.ocf.co.uk/, 2010. [Online]. Available: http://www.ocf.co.uk/

[10] "IBM – enCORE HPC on-demand service," http://www-304.ibm.com/partnerworld/gsd/scsolutiondetails.do?cd=BPAS&sbcd=&solution=44954&lc=en.

[11] T. J. M. H., T. R. P., A. R. J., D. M. T., A. K. F., W. A. M., B. R. P., P. L., and D. M. C., "Science carried out as part of the NW-GRID project using the eMinerals infrastructure," 2007, pp. 220–227. [Online]. Available: http://www.allhands.org.uk/2007/proceedings/papers/892.pdf

[12] 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 John Vincent Atanasoff Conference*. Los Alamitos, CA: IEEE CS Press, 2006, pp. 46–51.

[13] 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*, vol. 4, no. 1, pp. 9–16, 2008.

[14] K. Wasielewska, M. Drozdowicz, M. Ganzha, M. Paprzycki, N. Attaui, D. Petcu, C. Badica, R. Olejnik, and I. Lirkov, "Trends in Parallel, Distributed, Grid and Cloud Computing for engineering," P. Ivanyi and B. Topping, Eds. Stirlingshire, UK: Saxe-Coburg Publications, 2011, ch. Negotiations in an Agent-based Grid Resource Brokering Systems.

[15] M. Drozdowicz, M. Ganzha, K. Wasielewska, M. Paprzycki, and P. Szmeja, "Using ontologies to manage resources in grid computing: Practical aspects," in *Agreement Technologies*, ser. Law, Governance and Technology Series, S. Ossowski, Ed. Springer Netherlands, 2013, vol. 8, pp. 149–168.

[16] M. Drozdowicz, K. Wasielewska, M. Ganzha, M. Paprzycki, N. Attaui, I. Lirkov, R. Olejnik, D. Petcu, and C. Badica, *Ontology for Contract Negotiations in Agent-based Grid Resource Management System*. Stirlingshire, UK: Saxe-Coburg Publications, 2011.

[17] K. Chmiel, D. Tomiak, M. Gawinecki, P. Kaczmarek, M. Szymczak, and M. Paprzycki, "Testing the efficiency of jade agent platform," in *ISPDC/HeteroPar*, 2004, pp. 49–56.

[18] K. Chmiel, M. Gawinecki, P. Kaczmarek, M. Szymczak, and M. Paprzycki, "Efficiency of jade agent platform," *Scientific Programming*, vol. 13, no. 2, pp. 159–172, 2005.