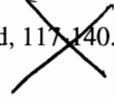




©Saxe-Coburg Publications, 2009.

Parallel, Distributed and Grid Computing for Engineering,
B.H.V. Topping and P. Iványi, (Editors),
Saxe-Coburg Publications, Stirlingshire, Scotland, 117-140.



Chapter ~~7~~

PROOF

Ontologies, Agents and the Grid: An Overview

PROOF

**M. Drozdowicz¹, M. Ganzha¹, M. Paprzycki^{1,2}, R. Olejnik³,
I. Lirkov⁴, P. Telegin⁵ and M. Senobari⁶**

¹*System Research Institute Polish Academy of Sciences, Warsaw, Poland*

²*Warsaw Management Academy, Warsaw, Poland*

³*University of Sciences and Technologies of Lille, France*

⁴*Institute for Parallel Processing, Bulgarian Academy of Sciences, Sofia, Bulgaria*

⁵*SuperComputing Center Russian Academy of Sciences, Moscow, Russia*

⁶*Tarbiat Modares University, Tehran, Iran*

PROOF

Abstract

One of the important claims that permeate the current view of information management is that ontological demarcation of data and semantic information processing are going to allow us to infuse “intelligence” into information systems. Separately, it is claimed that software agents, combined with ontologies will be the foundation of Web 4.0. In our work we are developing an agent-team-based resource management and brokering infrastructure for computational grids. The proposed meta-level middleware is to utilize both software agents and ontologies. In this context, the aim of this chapter is twofold. First, we present an overview of found efforts to develop ontologies to be used in grid and agent-grid computing. Second, we analyze which one of them, if any, should be the base ontology for the system under development.

Keywords: software agents, grid, resource brokering and management, ontology, semantic information processing.

PROOF

7.1 Introduction

Computational grids are considered one of more promising approaches to the development of a global computational infrastructure. Unfortunately, while a continuous stream of research funding is flowing into grids, their uptake is not meeting expectations of their proponents. One of possible reasons is the learning curve of a potential user, who tries to find out how to use the grid. In [1] it has been claimed that utilization of software agents may help the grid reach its potential. The metaphor here is that while the grid will provide the resources (data and computational power—the brawn), software agents will supply the system with “intelligence” (the brain). Obviously, this claim is somewhat controversial (*e.g.* because software agents have a number of staunch critics and/or those who claim that there is no such thing as a software agent, except for marketing [2]). Separately, according to Hendler [3], software agents go hand-in hand with ontologies and semantic data processing. Furthermore, in [4] one can find an argument that software agents and ontologies are going to be the foundation of Web 4.0. Obviously, these claims also need to be taken with a grain of salt, but the question arises, are the proponents or the critics of agents and ontologies right? It is our belief that the proof of the pudding is in the eating. One cannot validate either of these claims without an all out effort to make them work. This is the context of our current work. Assuming that software agents are one of the keystones for the development of the Grid and that ontologies and semantic data processing matches well with utilization of software agents we proceed the developing a meta-level agent-based grid resource management and brokering infrastructure.

In the proposed approach (the *Agents in Grids*; *AiG* project) software agents work in teams, which allows us to establish service level agreements (SLA) and assure quality of service. Each team is represented by an *LMaster* agent (its manager). The *LMaster* is supported by an *LMirror* agent, which helps facilitate long-term stability of the team (by mirroring all of the vital team data). Users negotiate with *LMasters* terms of job execution (the SLA), or team joining. Information about all existing teams, their work and job offers is stored in, and provided by, the *Client Information Center* infrastructure (represented by the *CIC* agent). It is assumed that all information in the system will be ontologically demarcated and semantically processed. A summary of the results obtained thus far can be found in [5–8], while the current version of the code is available at [9].

It should be noted that, initially, after a brief overview of grid ontologies, we have decided to develop and use a very simple one [8, 10]. For instance, the description of a grid node utilized only three of its possible features: (1) CPU specification, (2) available memory, and (3) available disk space. However, since our project has reached its initial goals, it becomes necessary to infuse the proposed system with a robust/-complete grid ontology. Therefore, in this chapter we provide an overview of existing efforts in the area of grid ontologies. Furthermore, we evaluate them from the perspective of their possible utilization as a core ontology of our system. Let us start our overview from resource descriptions found in core grid projects.

PROOF

7.2 Traditional grid resource descriptions

7.2.1 Virtual grid description language

The virtual grid description language (vgDL; [11, 12]) is a language for describing grid resources and querying grid information services. It was developed as a part of the VGrADS (Virtual Grid Application Development Software) Project [13]. The project itself is a result of collaboration between several American universities and is aimed at creating software systems and frameworks to simplify development of grid applications. VGrADS works together with several research projects and scientific applications to acquire practical use cases, but its deliverables have not (thus far) gained significant adoption.

The vgDL is built on the concept of *virtual grids*—a set of abstractions expressing the structure of the grid application. These abstractions are used to enable discovery of grid resources satisfying the specific criteria without the need to use concrete and overly detailed constraints. From the outset, the vgDL was designed to keep the insignificant details out of the way of the grid user. Let us stress that the vgDL does not serve as a schema for describing resources—it is rather a language to describe requirements and constraints that the user imposes on grid resources (the *Virtual grid*) to complete its task. In this aspect the vgDL is closer to the UNICORE AJO model than to the GLUE schema (see subsequent Sections).

On the most basic level, the requirements for a single resource in the vgDL are specified by a series of simple attribute value constraints. However, what makes vgDL powerful is the notion of resource *aggregators*:

- LooseBagOf—a set of heterogeneous resources connected in a loose way (*e.g.* with 'poor' connectivity),
- TightBagOf—a set of heterogeneous resources connected in a tight way ('good' connectivity),
- ClusterOf—a set of homogenous resources connected tightly.

These *aggregators* specify also the minimum and the maximum number of nodes in the set and the constraints put on the nodes. Furthermore, they can be nested to form hierarchies of nodes.

Another key concept of the vgDL is support for specifying the network connection type between nodes and aggregates. This is achieved through the use of four network connectivity operators: (i) close, (b) far, (c) lowBW, and (d) highBW that describe latency and bandwidth.

These features provide users with a way to easily specify even complex needs for computing and storage nodes; taking into account the physical and geographical architecture of the required environment.

Disadvantages of this approach, with regard to its usefulness in the *AiG* project are:

- Focus on describing requirements and constraints, while missing the description of the resource itself—no specification of how the information about resources

should be stored.

- Lack of specification of a standard set of attributes and their values, and/or their mapping to existing grid Information Services.
- vgDL does not explicitly handle the description of requirements regarding installed software or libraries—examples of how this may be implemented use values of custom attributes.
- vgDL is not adopted in core grid projects and was verified only in experimental conditions.

Bearing this list in mind, we decided not to investigate the vgDL project further.

7.2.2 Condor

The Condor grid middleware [14–17] is a high-throughput computing system, providing users with an environment for transparent execution of jobs on multiple, geographically distributed machines. It can run non-interactive jobs on any node under control of the system without the need to modify the source code. The user simply submits a job, and Condor handles matching the job to available machine(s) and executing it. If a node becomes unavailable, Condor can automatically backup the execution state of the job and migrate it to another machine satisfying its requirements. Condor supports single-process and parallel applications written using different MPI implementations. Furthermore, jobs submitted to Condor can be ordered to enable sequential execution of dependent tasks.

All entities within Condor are described using the same abstract language — Classified Advertisement (ClassAd; [18, 19])—that can serve both as a means of describing elements of the system and as a query language. ClassAd has been designed to enable efficient and extensible matchmaking of resources to requirements. Providers of resources can offer them to the system by submitting ads describing their capabilities and constraints regarding jobs they can run. The same language is used by consumers of resources for describing their needs, requirements and preferences with regard to hosts their job should be run on, as well as for describing the job itself. ClassAd is a functional language built around the concept of an *expression* evaluated by the engine. The core expression type in ClassAd is the *record expression*, which is basically a set of name-value pairs, with the value being either a simple literal or a complex expression, including nested records.

Matchmaking of ClassAd job-machine pairs is done based on the definition of an attribute called *requirements*. The value of this attribute can be an arbitrary expression that is evaluated in the environment of the other record expression. Therefore if a job is to be matched to a machine, the job's Requirements must evaluate to true against the machine's description record, and the machine's requirements in the job's description record.

The Condor manual [20] provides a list of machine description attributes along with their valid values. This list is very extensive and contains attributes sufficient for most common usages of the system. The built-in attribute set does not, however, allow the job to have requirements concerning specific libraries or software installed or running

on the host machine. In such scenarios, machines satisfying these requirements would need to create and advertise this information as custom attributes. Furthermore, Condor cannot perform advanced matching by following rules defining the relations between certain attribute values such as a hierarchy of operating system types and their concrete versions (*e.g.* all Linux kernel version share the same value of the *OpSys* attribute).

Looking into usability of the ClassAd language in our project, we conclude that the set of attributes describing a machine could certainly be included in the needed ontology. On the other hand, the ClassAd language seems too proprietary and restricted in defining ontological metadata of system resources and relationships between elements used to describe them. Therefore we cannot consider it as a strong candidate to be used in our system.

7.2.3 UNICORE

The motivation for creating another popular grid middleware—UNICORE (Uniform Interface to Computing Resources [21])—was completely different to that of Condor. Condor was primarily designed to take advantage of desktop computers and small machines idle processor time and to enable executing jobs in a dynamic environment. UNICORE's aim, on the other hand, was to provide (originally German) supercomputer users with uniform access to various resources (scattered around the country) and to make job scheduling easier and infrastructure independent. In its core version the user would actually select a single target on which the job should execute. In contrast to the Condor, UNICORE on its own does not provide resource brokering, job checkpointing or migration.

Jobs in UNICORE are described at the user tier of the system using Abstract Job Objects (AJO)—a collection of Java classes that contain all information needed by the UNICORE target tier to prepare a native batch for execution. Therefore, UNICORE does not provide a “language” for describing jobs—it relies on plain Java objects that are serialized and sent to the UNICORE server. Furthermore, users that do not want to use the provided GUI for job submission can use the UNICORE client API to create a job description programmatically. Resource descriptions are stored in the *Incarnation Database* (IDB) which contains information needed to transform the abstract job to a task ready for execution on a specific grid resource. One of the serious drawbacks of the IDB was that it contained only static information—software and hardware resources. UNICORE did not provide resource monitoring capabilities and therefore did not support dynamic grid utilization.

Unfortunately for our project, UNICORE does not really define a resource ontology. Its main focus is on describing jobs that need to be executed and on how to translate these descriptions in a specific environment where the execution is to take place. Some of the deficiencies of the original UNICORE were fixed along with the changes in the schemas of the AJO and the IDB in UNICORE version 6, thanks to the UniGrids project, but this topic will be considered separately in section 7.3.4.

7.2.4 Globus Toolkit

The Globus Toolkit [22] is a set of libraries and programs designed to support creating distributed applications. It addresses such problems as resource access, resource management, service discovery, security, data transfer and migration, *etc.* The core of the Globus Toolkit is not so much of a complete system for executing computation jobs (as was the case with Condor or UNICORE) but rather a foundation on which more specialized services are built. Over the years, Globus Toolkit evolved rather noticeably and in the fourth version its core functionality and interfaces are primarily based Web Services.

Considering the resource discovery and management, along with new versions of Globus Toolkit came also a new version of Globus's implementation of the grid Information Service—the Meta Directory Service. Globus Toolkit 1 used MDS-1 which relied on a single LDAP server which held information about all resources in the system. Resources would sign up into the server and then periodically send updates of their state. Users would utilize a single server to query the state of a given resource as well as to find nodes satisfying specific requirements. This solution's main problem was poor scalability—LDAP was optimized for a number of queries but not for frequent updates, which occurred when resources sent information about their load. In MDS-2 [23] the central server's role was limited to responding to discovery queries, while finding information about a specific resource was moved to the resource itself. Here, grid resources started carrying their own LDAP servers (GRIS) that contained information about their state. The server (GIIS) responded to discovery queries by sending information requests to all resources and only then updating its internal cache of resource descriptions.

The information about resources contained in the LDAP databases was structured following the schema defined in [24]. Specifically, it was organized around a small set of elementary classes and a much wider collection of auxiliary classes that inherited the base *MDS* class and corresponded to concrete resource instances, *e.g.* the CPU, memory or file system. Properties of these resources were described using LDAP attributes. In the following snippet a description of memory resource is shown (example from [24]).

```
dn: Mds-Device-Group-name=memory, ...
objectclass: MdsMemoryRamTotal
objectclass: MdsMemoryVmTotal
objectclass: MdsDeviceGroup
Mds-Device-Group-name: memory
Mds-validfrom: 200110030128.12Z
Mds-validto: 200110030128.12Z
Mds-kepto: 200110030128.12Z
Mds-Memory-Ram-Total-sizeMB: 751
Mds-Memory-Ram-Total-freeMB: 642
Mds-Memory-Vm-Total-sizeMB: 1600
Mds-Memory-Vm-Total-freeMB: 1592
Mds-Memory-Ram-sizeMB: 751
```

PROOF



Mds-Memory-Ram-freeMB : 642

Mds-Memory-Vm-sizeMB : 1600

Mds-Memory-Vm-freeMB : 1592

Along with the introduction of GT3, the Information Services' LDAP databases were abandoned in favor of XML based information repositories. Moreover, the description schemas also underwent major redevelopment—the custom Globus solution was replaced by a more generic and interoperable GLUE schema, described next.

7.2.5 GLUE

GLUE (Grid Laboratory for a Uniform Environment) is a project started in April 2002 as a joint effort between the U.S.-based iVDGL and the European DataTAG teams. Its main purpose has been to bring interoperability between the U.S. and European physics grid projects, through a uniform schema specifying description of grid resources needed for implementation of grid Information Services, resource discovery and brokering systems. The first version of GLUE was released in October 2002 and contained definitions of a grid computing element and storage service. Since then, subsequent documents (1.1, 1.2, 1.3) have expanded the number of supported use cases, merged and unified notions of computing and storage element into a single schema, as well as included additional concepts, *e.g.* those related to administration of resources. Since October 2006, the GLUE project has joined the Open Grid Forum and its efforts, focused on finishing version 2.0 of the schema, are continued within the GLUE Working Group ([25]).


The schema was published as a description of the conceptual model of a structure of grid resources, complemented by UML Class Diagrams and tabular descriptions of defined entities and their relationships. Notions of the schema are independent of the technology used to enforce the schema and of data model languages. For the schema to be usable it needs to be mapped to concrete implementations. Such realizations exist for grid resource descriptions modeled in XML (XSD Schema), LDAP and SQL (data base structure) for both 1.3 and 2.0 versions of GLUE. Version 1.3 for XML was natively supported by the Globus Toolkit 3 and a mapping to Condor ClassAds also exists. The following summary of the schema is based on version 2.0.

The GLUE schema is built around a set of abstract entities that define the basic relationships between elements of the system. Users of the system are organized using a notion of a *Domain*—a set of actors. The *Domain* can be given access to services through the *Access Policy* entities. There are entities derived from the *Domain*: *User Domain*, which reflects a *Virtual Organization* able to access services; and *Admin Domain* with primary goal of hosting services. Both *User Domain* and *Admin Domain* can be organized in a hierarchical structure. *Services* are always accessible as one or several *Endpoints* and the *Access Policies* actually define rules of accessing *Endpoints*. A *Service* is composed of *Managers*—entities managing sets of *Resources*—and *Shares*—targets for sets of resources exposed through *Endpoints*. A *User Domain* submits an *Activity* using an *Endpoint*. The *Activity* is run on a *Share* into which the *User Domain* is mapped by the appropriate *Mapping Policy*.

The schema specifies implementation of defined abstract entities—the *Computing Service* and the *Storage Service*. Because of the current interest of the AiG project we will omit the *Storage Service* and focus on the grid computing resource description. The *Computing Service* reifies the abstract concepts of the generic *Service* in the following way:

1. The concrete implementation of the *Manager* is the *Computing Manager*, which in the simplest case is just the operating system controlling a single node. In the typical case this might be a batch system (Local Resource Management System). In a more complex scenario, this could also be a meta level middleware such as the Condor.
2. The *Computing Activity* is an OGSA compliant description of a job submitted to the *Computing Service* through one of its *Computing Endpoints*.
3. The *Computing Manager* manages a set of *Execution Environments* that depict physical or virtual computational resources of the Grid. The *Execution Environment* description is the element that contains the hardware, software and network characteristics of the resource. The *Execution Environment* contains data properties such as (for a complete description including types of properties, see [26]):
 - type of the platform the machine runs on,
 - number of physical CPUs in an instance of the *Execution Environment* (one physical CPU per socket),
 - number of logical CPUs (*i.e.* the number of CPUs seen by the OS) in an instance,
 - detailed CPU information, such as the name of the vendor of the CPU, the model, version and clock speed,
 - amount of physical (RAM) and virtual (RAM + swap) memory,
 - information about the operating system such as family, name and version,
 - type of the network connection between the instances composing the *Execution Environment*.

Apart from the system specification, GLUE also specifies a way of describing available software. This is achieved through the use of *Application Environment* entities and their relationships with the *Execution Environment*. In the XSD snippet below, we have shown a fragment of the XML Schema describing this part of the specification of the *Execution Environment* [27]. We can see that the *ExecutionEnvironment_t* type is inheriting the *Resource_t* type. It consists of a number of optional (minOccurs="0") properties describing the platform, number of instances of the environment, number and details of instances' CPUs, memory, details of the operating system and network connectivity. The type definition ends with a list of relations to other types (the *Associations element*) such as the *Computing Share*, *Application Environment* and *Computing Activity*.

A large, stylized, outlined stamp with the word "PROOF" in all caps, slanted upwards from left to right, positioned in the upper right quadrant of the page.

```
<complexType name="ExecutionEnvironment_t">
  <complexContent>
    <extension base="glue:Resource_t">
      <sequence>
        <element name="Platform" type="glue:Platform_t"/>
        <element name="VirtualMachine"
          type="boolean" minOccurs="0"/>
        <element name="TotalInstances"
          type="unsignedInt" minOccurs="0"/>
        <element name="UsedInstances"
          type="unsignedInt" minOccurs="0"/>
        <element name="UnavailableInstances"
          type="unsignedInt" minOccurs="0"/>
        <element name="PhysicalCPUs"
          type="unsignedInt" minOccurs="0"/>
        <element name="LogicalCPUs"
          type="unsignedInt" minOccurs="0"/>
        <element name="CPUMultiplicity"
          type="glue:CPUMultiplicity_t" minOccurs="0"/>
        <element name="CPUVendor"
          type="string" minOccurs="0"/>
        <element name="CPUModel"
          type="string" minOccurs="0"/>
        <element name="CPUVersion"
          type="string" minOccurs="0"/>
        <element name="CPUClockSpeed"
          type="unsignedInt" minOccurs="0"/>
        <element name="CPUTimeScalingFactor"
          type="float" minOccurs="0"/>
        <element name="WallTimeScalingFactor"
          type="float" minOccurs="0"/>
        <element name="MainMemorySize"
          type="unsignedLong" minOccurs="0"/>
        <element name="VirtualMemorySize"
          type="unsignedLong" minOccurs="0"/>
        <element name="OSFamily"
          type="glue:OSFamily_t" minOccurs="0"/>
        <element name="OSName"
          type="glue:OSName_t" minOccurs="0"/>
        <element name="OSVersion"
          type="string" minOccurs="0"/>
        <element name="ConnectivityIn"
          type="boolean" minOccurs="0"/>
        <element name="ConnectivityOut"
          type="boolean" minOccurs="0"/>
        <element name="NetworkInfo"
          type="glue:NetworkInfo_t" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

```

<element name="Extensions"
  type="glue:Extensions_t" minOccurs="0"/>
<element name="Benchmark" type="glue:Benchmark_t"
  minOccurs="0" maxOccurs="unbounded"/>
<element name="Associations" minOccurs="0">
  <complexType>
    <sequence>
      <element name="ComputingShareLocalID"
        type="string" minOccurs="0"
        maxOccurs="unbounded"/>
      <element name="ApplicationEnvironmentLocalID"
        type="string" minOccurs="0"
        maxOccurs="unbounded"/>
      <element name="ComputingActivityID"
        type="glue:ID_t" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
</sequence>
</extension>
</complexContent>
</complexType>

```

With regard to the *AiG* project, GLUE is interesting and helpful in at least a few ways. One aspect is the approach to the role of the grid resource description—it is primarily the input to the resource directory and brokering services. Another important advantage of the GLUE schema is its wide adoption among grid projects and native usage in the Globus Toolkit 4. This comes together with the maturity of the schema and its covering of a large number of real world-use cases. A natural way of utilizing the schema would be reusing definition of the *Computing Service* element. In this case a huge advantage of GLUE is its well-defined, object model as well as the ability to model not only physical specifications of the resources, but also available applications and libraries. One clear disadvantage of reusing the GLUE schema is lack of any ontological (*e.g.* RDF/OWL [28, 29]) realization. Thus, it would be necessary to create one from scratch. Another is the fact that while GLUE is compatible with many grid middlewares it still does not support *UNICORE*'s approach. Therefore, within the context of the *AiG* project, we are likely to use the GLUE schema as reference, especially in the area of description of the physical and virtual computing resources.

7.3 Semantic grid description and grid interoperability

Thus far we have looked into grid resource descriptions found among traditional grid efforts (which have never aimed at utilization of ontologies). Out of them we have

found that only the GLUE schema could be utilized as the foundation of an ontology needed in the *AiG* project. However, this would mean starting from scratch and creating an ontology using an ontology language (*e.g.* OWL) from existing XML based descriptions. Therefore, we will now focus our attention on efforts that already utilize ontological demarcation. Interestingly, many such projects originate from efforts to support greater heterogeneity among grid resources both in terms of attribute value domains and grid middleware interoperability (see, [30] for more details).

7.3.1 The grid resource ontology of Pernas and Dantas

An example of work towards an ontology of grid resources is found in [31]. Here, the authors' goal was to create an ontological intermediate layer between the consumer of grid services and grid resources. This was to enable easier and faster discovery of the resources as well as to provide common vocabulary for different VOs. The proposed ontology was implemented in the OWL Full language.

As far as we were able to determine, the project described in [31] has been completed in 2005 and since then the ontology has not been developed further. Furthermore, the developed ontology comprised merely 14 classes. This being the case it was actually less complete than the one used thus far in the *AiG* project. These reasons lead us to disregard this effort from our consideration.

7.3.2 The grid resource ontology of Vidal *et al.*

PROOF

The grid resource ontology described in [32] was created as an effort to support resource brokering in a grid middleware system. Here the focus of the project was to enhance resource matching processes. The proposed ontology was used by an additional intermediate layer between the client of the grid and the grid resource manager to analyze constraints specified by the client and transform them using automatic reasoning to match more resources, not found using traditional attribute-based matching. The ontology was based on the OWL DL language and consisted of three parts:

- *grid based ontology*—the foundation ontology consisting of basic concepts describing physical resources (Computer, Cluster, DiskSpace *etc.*) and software (Problem, Domain, Algorithm, Application)
- *platform ontology*—an ontology extending the platform related concepts of the *Base Ontology* with the hierarchy of CPU architecture and operating systems
- *grid resource management ontology*—a domain ontology specifying problems and algorithms related to data mining.

An important aspect of this ontology and its usage is that it is *not* used as a complete description of the resource or its constraints, but rather as a set of concepts useful for describing attribute values and enabling reasoning about them. The result of such an approach is that relations contained in the ontology describe the equality or compatibility of concepts. For example in the *Platform Ontology* there is a property stating that the *Itanium processor* class has architecture of type *ProcessorArch64Bits*. There

is, however, no property stating that the *Computer* class is related to the *Processor* class. The ontology on its own does not contain data properties such as the amount of available memory or the speed of the processor. It also cannot be used to describe jobs to be submitted to the Grid.

A complete set of ontology files can be downloaded from [33] but judging from publications related to the project, the ontology itself has not been actively developed since June 2007. Furthermore, the goal and purpose of this ontology seem to be different to the needs of our project and therefore we have decided not to pursue it further.

7.3.3 GRIP

PROOF

The GRid Interoperability Project (GRIP) [34] was aimed at bringing about interoperability between UNICORE and Globus grid middlewares as well as providing brokering capabilities for the UNICORE. It was a 2-year project running in 2002-04. Its results included creating interoperability layers between UNICORE and Globus Toolkit v2 and v3, an abstract ontology of the resource, and a common resource broker for both Globus and UNICORE resources. The project also laid foundations for the adoption of the OGSA standards in UNICORE.

From the point of view of the aim of this chapter, the most interesting aspect of GRIP is the matching and mapping phase of creating the Globus-UNICORE intermediate layer. Since the process of creating a translator between the two has been described in a rather detailed way in [34], below we highlight only the most important points. The resource ontologies for both systems were extracted using the PCPack tool [35] from the GLUE schema in case of Globus and from the AJO JavaDoc documentations for the UNICORE. While the PCPack does not natively support the OWL language, it enabled us to easily, graphically customize and refine resulting ontologies. Afterwards the two ontologies were compared to try to match concepts contained therein. The most important differences found in the process were those of the scope of descriptions. UNICORE descriptions contained information about static hardware and software capabilities of the resource. The GLUE schema, on the other hand, did not specify software capabilities. However, apart from static hardware capabilities, it described the dynamic information, *e.g.* the current load of the resource. This being the case, only the intersection of these two “universes,” *i.e.* the static description of hardware capabilities could have been mapped.

Unfortunately we were not able to find the complete PCPack ontologies used in the course of the project, nor could we acquire the XML files used for mapping the ontology terms. Therefore, we cannot present an example of how these would describe a sample resource or discuss details of the implementation. We can, however, discuss results of two projects following this effort—the uniform interface to grid services (UniGrids) and the Grid Ontology project.

7.3.4 UniGrids, OGSA and the grid ontology

UniGrids (Uniform Interface to Grid Services) was a follow-up on the GRIP project. Its main goal was to design and implement a grid middleware system compliant with the Open Grid Service Architecture (OGSA) standards. It was to be based on the UNICORE and the results of the GRIP project. However, as the project developed, it was substantially modified—especially looking into the internal structure of the proposed ontology.

In terms of resource descriptions and requests, the UniGrid project brought the adoption of the XML to the description of the incarnation process. Prior to the UNICORE 6, the incarnation process was described using custom formatted text files, while in version 6 they became XML-formatted and contained elements from the OGSA JSDL namespace. These were used mostly to describe the resource capabilities description such as the operating system, number of nodes or CPUs per node. Furthermore, the UniGrids project brought the specification of the grid Resource Ontology and the more general grid Ontology [36]. The ontology was designed to form a basis for automatic matching of many different grid middlewares as well as an easy addition of new systems.

The proposed ontology was built using the OWL language and was split into several layers. The core layer (referenced in [36] as the *foundational ontology*) is a set of classes forming the high-level, common view of grid elements, such as virtual organizations, abstract resources, security related entities, abstracts actions and tasks. On top of the base ontology, the OGSA and the S-OGSA ontologies were defined. They introduce concepts related to these standards as well as middleware specific ontologies such as the Globus Ontology and the UNICORE ontology.

A very important part of the ontology are classes and object properties representing the abstract grid job and the way it is translated to concrete processes for the target machine. Approach presented in the ontology is very similar to the one from the UNICORE—it is based on the concept of the *AbstractJob* and the idea of its *Incarnation* to the local resource. An *AbstractJob* can also contain multiple *Actions* so it can be used to model workflows. Apart from jobs, the process of incarnation also covers concepts such as the file system *Directories*, the system *Processes* and user logins. The ontology also contains a set of classes related to the *quality of service* and *service level agreements* and their negotiation. This functionality is provided by means of the *QoS* class and its subclasses: *Advertised_QoS*, *Agreed_QoS*, *Observed_QoS* and *Requested_QoS*. The *Quality of Service* is defined on the level of single *Actions* and *Resource_Sets* assigned to them and negotiated by the *Broker*. It should be noted that while these classes are joined with relations to other concepts of the ontology, they lack data properties describing the concrete terms of the agreement, such as for example: start time, end time, and cost of running the *Action* on the *ResourceSet*. Note that these concepts are used directly in our system when users negotiate jobs with *LMasters* [8].

Looking into more details, resources are described using the *Resource* abstract entity, which is inherited afterwards by more specific resource type entities: *Disk*, *File*,

Network, Processing, Resource Collection (which basically is a set of resources) and *Software*. Properties of the ontology provide means of describing resource structures of arbitrary complexity combining computing and storage capabilities. Properties of resources are specified by the *Resource_Property* entities which correspond to the WSRF WS-ResourceProperties standard. To illustrate the approach, in Figure 7.1 we can see the resource hierarchy along with the way they are exposed to clients—through the *Service* class.

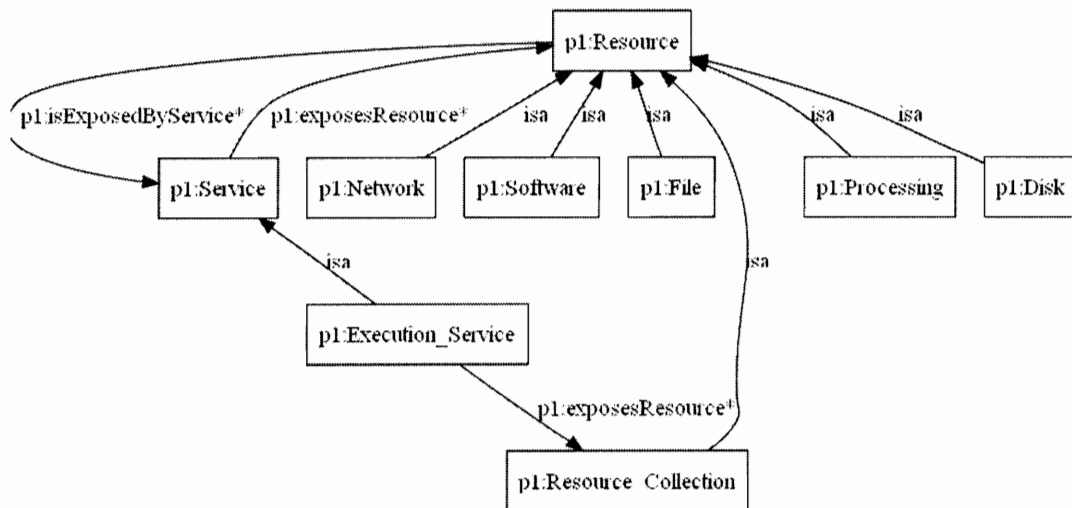


Figure 7.1: The hierarchy of resource concepts in the grid ontology

The important thing to note is that the WS-ResourceProperties standard does not define a set of properties that can be used for describing resources. Instead it defines a meta-layer of these properties—it includes terminology, concepts and operations needed to expose the definition of properties and their values; query and update the property values. Therefore, the ontology itself does not provide concepts needed to completely describe the physical characteristics and capabilities of the resource. As noted above, it also does not provide us with the right set of concepts needed for job contract negotiations. We find this to be a serious deficiency as it would require a significant effort to extend what exists to the point of usefulness in our project. On the other hand, we can see here a mature and complete set of concepts describing the overall structure of the grid infrastructure as well as the classes related to abstract job descriptions, which might provide us with very important ideas and insights.

7.3.5 Core grid ontology

Finally, let us describe the effort that seems to be the closest to our needs and that is likely to be adopted in our project. The *Core grid Ontology* ([37], [38]) is an OWL demarcated, grid architecture independent, collection of concepts and relations between them that could be extended to include middleware specific properties. In this case, the authors did not create an ontology with any specific goal (*e.g.* ontology

of a specific middleware) but instead wanted to provide a generic ontological grid description for further extension and use by semantic grid applications.

Proposed ontology does not capture the whole universe of the Grid but was created by examining various grid architectures and extracting core concepts common to all of them. The result of this process is a layered model of the generic grid architecture, containing, looking from the top to the bottom:

- *grid VOs and applications*—classes such as *VO*, *GridUser*, *GridApplication* and *Policy* describing the configuration and application of the Grid,
- *grid middlewares and services*—grid middleware, components, libraries and services enabling core grid functionality,
- *Resources*—classes describing resources, such as *ComputingElement*, *WorkerNode* and various resource physical description concepts like *CPU* or *StorageInterface*.

Apart from the classes, the proposed ontology also defines relations between them, using object and data properties. Thanks to these properties, it is possible to create a complete structure of a multi-VO grid environment with interconnected elements. The *VO* contains registered participants - either *GridResources* or *GridUsers* and can contain *Sites*. Each *Site* can have *GridComponents* such as *ComputingComponent*, *StorageComponent* or *ResourceBroker*. The important class for computing jobs is the *ComputingElement*—a subclass of *ComputingComponent*—which can have *Queues* and *WorkerNodes* as well as installed software such as *GridMiddleware* and *OperatingSystem*. The *WorkerNode* is the resource which runs the job, and which is likely to contain a *CPU*. The vocabulary of concepts representing grid resources is also rather extensive—we have shown it in Figure 7.2.

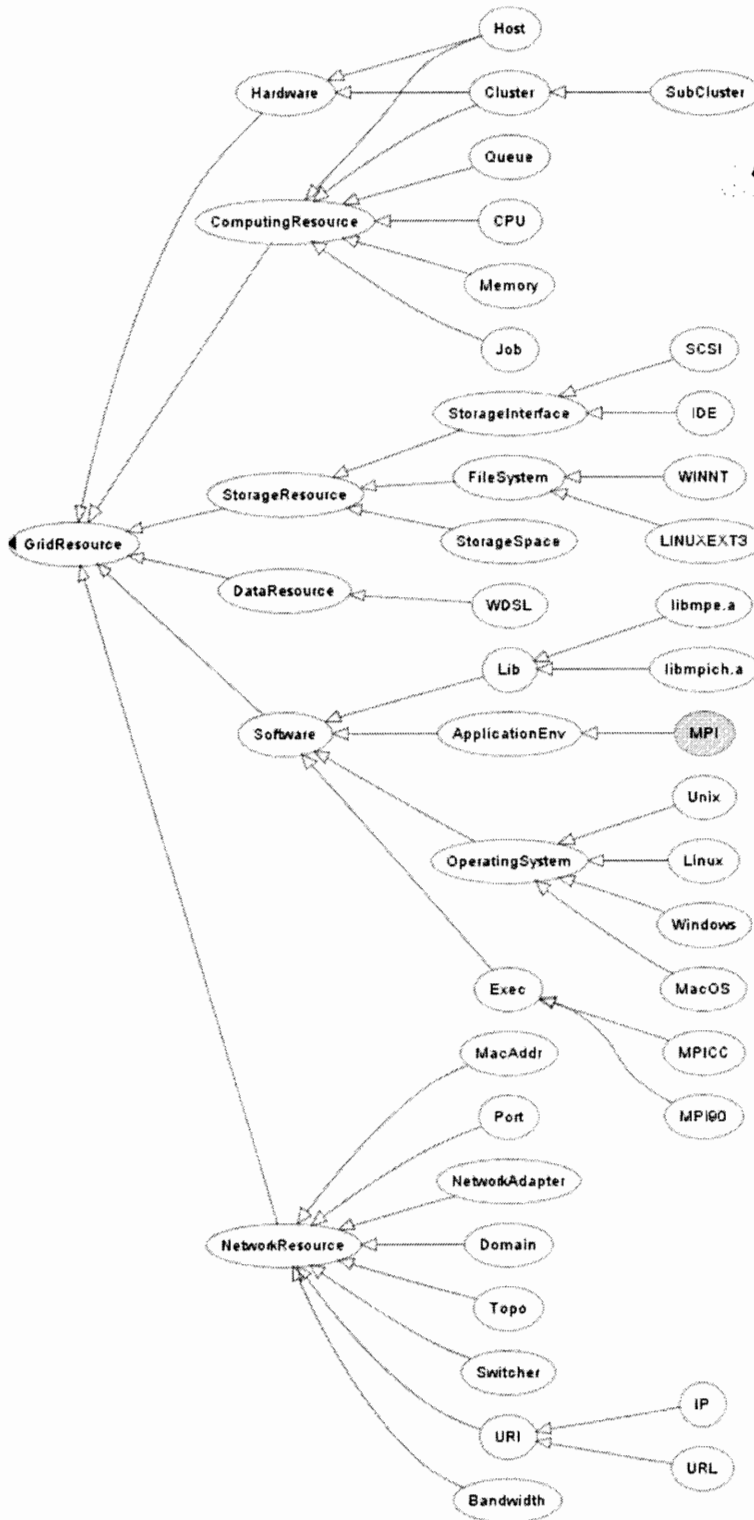
Figure 7.3 shows a diagram of the structure of the sample grid site—that of the University of Cyprus Grid—based on the instances provided in the ontology OWL file. We can see that the CY01-LCG2 site has four *GridComponents*:

- *ui101.grid.ucy.ac.cy*—an instance of the *UI* class
- *se101.grid.ucy.ac.cy*—a *StorageElement*
- *rb101.grid.ucy.ac.cy*—a *ResourceBroker*
- *ce101.grid.ucy.ac.cy*—a *ComputingElement*

The *ce101.grid.ucy.ac.cy* computing element, in turn, handles three *WorkerNode* instances, the *dteamQ Queue* and runs two services: *openpbs_ucy* which is an instance of the *PBS* job manager and *maui_ce_ucy*—an instance of the Maui job scheduler.

Worker nodes are described as shown in the following OWL snippet.

```
<WorkerNode rdf:ID="wn102.grid.ucy.ac.cy">
  <belongsToVO>
  <VO rdf:ID="Dteam">
    <hasName rdf:datatype=
      "http://www.w3.org/2001/XMLSchema#string">
      dteam
```



PROOF

Figure 7.2: The hierarchy of grid resource concepts in the Core Grid Ontology

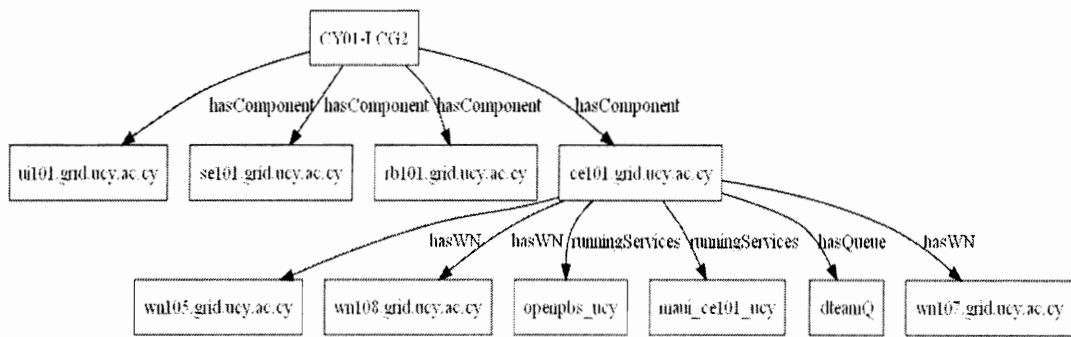


Figure 7.3: The structure of a sample grid site described using the Core grid Ontology

```

    </hasName>
  </VO>
  </belongToVO>
  <hasID>
  <IP rdf:ID="IP_WN102_UCY">
    <hasName xml:lang="en">194.42.17.237 </hasName>
  </IP>
  </hasID>
  <hasCPU>
  <CPU rdf:ID="CPU_Intel">
    <availableNum
    rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      2
    </availableNum>
    <hasModel xml:lang="en">Xeon</hasModel>
    <clockSpeed xml:lang="en">2.8GHz</clockSpeed>
  </CPU>
  </hasCPU>
  <hasName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
  </hasName>
  <InstalledSoftware>
    <LCG rdf:ID="LCG_2.6.0"/>
  </InstalledSoftware>
  <InstalledSoftware>
    <Linux rdf:ID="Scientific_Linux_303">
    <rdfs:comment
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      A linux OS.
    </rdfs:comment>
    </Linux>
  </InstalledSoftware>
</WorkerNode>

```

As can be seen from the above description, relationships between classes are relatively complete, although there are still some missing concepts; e.g. lack of properties related to many of the *gridResource* subclasses such as the *NetworkResource*, *Memory* or *StorageResource*. However, the collection of grid infrastructure related classes is rather impressive—containing entities such as *GridInformationService*, *JobManager*, *JobScheduler* or *WorkloadManagement*.

A very helpful element of the ontology is the provided example set of instances of the classes corresponding to authentic elements of a grid environment in the EGEE project from which we have taken the examples shown in this section.

PROOF

7.4 Agent—grid integration

Thus far we have looked into traditional grid resource descriptions and attempts at developing a grid ontology. However, since our project is going to utilize both ontologies and software agents, let us now focus our attention on efforts to provide an ontological basis to agent-grid integration efforts. Here we discuss two approaches that we were able to identify.

7.4.1 The Agent Computational Grid

The first example is the *Agent Computational Grid (ACG)* project [39,40]. Motivation for this effort was to propose an agent-based computing and service exchange environment in which agents expose selected core features of the grid functionality; e.g. member registration and service discovery. Proposed agents do not introduce significant cognitive features to the grid (are not an actual “brain”), but utilize capabilities such as environment awareness and adaptability to improve the robustness of the grid and enable grid functioning under highly dynamic conditions. Another concept of the *ACG* is interoperability of grid members. It is suggested that the system should be able to accept agents of different platforms as well as enable exposing legacy services. Here, agent interoperability is accomplished on the basis of the *Mobile Agent System Interoperability Facility* standard [41], while wrapping of legacy systems is based on the integration of *CORBA* distributed objects technology. The *ACG* is designed around two way *CORBA* integration—existing *CORBA* services can be exposed through an agent based interface at the same time agents’ capabilities can be used as *CORBA* services.

Services in the *AGC* are described by means of XML with a clearly defined DTD schema. As specified in [40] the description contains the following fields:

- *Name*: This is an optional string that can be associated with a service
- *Service Id*: This is a required element that uniquely identifies the service. Each service has a unique id
- *Description*: describes some features of this service
- *Access*: describes how to invoke and use a service The access method fields provide the placeholder for any kinds of service invocation schema. A service

description may contain multiple access method tags as there could be multiple ways to invoke a service. The intent is for the agent to ultimately decide which invocation mechanism to use. Some agents may want to use a interface of services, some may decide to download a client-proxy code, another agent may migrate to destination node, and access the services locally

- *Interface*: indicate agent can access service by a standard interface
- *URL*: gives location of services supplied by a legacy system
- *Agent platform*: the agent platform on which the application was built
- *Service type*: informs the user of the type of service
- *Properties*: denotes a list of properties of the service.

Looking at the list of service description elements, it can be noted that the authors concentrated mostly on the aspect of multi platform interoperability, while leaving the issue of describing the execution environment and grid node resource capabilities practically untouched. In fact the only place where one could introduce information such as the physical specifications of the machine the service is hosted on, or the available system resources, or libraries, would be the *Properties* element. Unfortunately for this element, the authors did not provide any list of valid child elements. Therefore, even though the interoperability-focused approach to agent-grid integration proposed within the ACG project seems interesting in many aspects, the grid resource description has proved to be insufficient to our needs. Furthermore, despite our best efforts, we could not find traces of this project being extended past 2005.

PROOF

7.4.2 AGIO

The Agent-Grid Integration project, described in [42–44], is build around concepts of *agent* and *service*. In the proposed model, artificial or human *agents* are engaged in exchange of services and, for shared access to resources, are composed into *virtual organizations*. Integration between the agent and the grid worlds takes place on the level of services, *i.e.* services interface agents' capabilities using the concept of a *Cognitive Environment*. This feature enables grid services to behave in a completely stateful manner; not only for the duration of fulfilling a single request but also over the course of several user conversations. Moreover, thanks to the *Cognitive Environment* it would be possible to reason about the outcome of service calls and enrich results returned to clients.

In terms of formalization of the model, this effort has first been modeled using a specially created diagram language called the *Agent-Grid Integration Language (AGIL)*; described in [42]. The *AGIL* language enables a graphical representation of an agent-grid integrated system along with the relationships between its elements. The next step in the formalization of this concept was the *Agent grid Integration Ontology* as presented in [44], which contains *OWL* demarcated ontologies of all *AGIL* concepts as well as the *SWRL* ([45]) rules enabling machine parsing of the descriptions and automatic validation of model instances.

To summarize classes forming the ontology we will start with the concept of an *Agent*. *Agents*, organized into *VOs* and identified by the *X509* certificates, interact with

each other and use or provide *Services*. An *Agent* providing a *Service* can dedicate its *CognitiveEnvironment* to its clients. *Services* are included in *ServiceContainers* and handled by a *CAS* (Community Authorization Service). As mentioned earlier, *Services* interface agents' *Capabilities*. The *ServiceContainer* reifies a set of *VirtualizedResources* which in turn can virtualize a number of *Hosts*, that can be composed of a *ComputingResource* and a *StorageResource*. This ontology does not describe detailed properties of the *Resource* classes, nor does it specify the software or libraries accessible to the service clients. In the following XML snippet, we present a sample set of instances forming a single service.

```

<Capability rdf:ID="ComputingCapability">
  <isInterfacedBy>
    <ServiceContainer rdf:ID="ComputingServiceContainer">
      <interfaces rdf:resource="#ComputingCapability"/>
      <includes>
        <NormalService rdf:ID="ComputingService">
          <isIncludedIn rdf:resource="#ComputingServiceContainer"/>
          <interfaces rdf:resource="#ComputingCapability"/>
          <isHandledBy>
            <CAS rdf:ID="ComputingCAS">
              <isIncludedIn rdf:resource="#ComputingServiceContainer"/>
              <isHandledBy rdf:resource="#ComputingCAS"/>
              <handles rdf:resource="#ComputingService"/>
              <handles>
                <NormalService rdf:ID="SomeService">
                  <isHandledBy rdf:resource="#ComputingCAS"/>
                  <isIncludedIn rdf:resource="#ComputingServiceContainer"/>
                </NormalService>
              </handles>
              <handles rdf:resource="#ComputingCAS"/>
            </CAS>
          </isHandledBy>
          <isProvidedBy rdf:resource="#ComputingAgent"/>
        </NormalService>
      </includes>
      <includes rdf:resource="#SomeService"/>
      <includes rdf:resource="#ComputingCAS"/>
      <reifies>
        <VirtualizedResource rdf:ID="VRComputingNode">
          <virtualizes>
            <SimpleHost rdf:ID="ComputingHost">
              <isComposedBy>
                <StorageResource rdf:ID="StorageResource_1">
                  <composes rdf:resource="#ComputingHost"/>
                </StorageResource>
              </isComposedBy>
              <isComposedBy>
                <ComputingResource rdf:ID="ComputingResource_1">
                  <composes rdf:resource="#ComputingHost"/>
                </ComputingResource>
              </isComposedBy>
              <isVirtualizedBy rdf:resource="#VRComputingNode"/>
              <holds>
                <X509Host rdf:ID="X509Host_11">
                  <isHeldBy rdf:resource="#ComputingHost"/>
                </X509Host>
              </holds>
            </SimpleHost>
          </virtualizes>
          <isReifiedIn rdf:resource="#ComputingServiceContainer"/>
        </VirtualizedResource>
      </reifies>
    </ServiceContainer>
  </isInterfacedBy>
</Capability>

```

```
</isInterfacedBy>  
<isInterfacedBy rdf:resource="#ComputingService"/>  
</Capability>
```

Here, we can see the *ComputingCapability*, which we assume enables the agent to run some client-specified job, is interfaced by the *ComputingService*. The *ComputingService* is included in the *ComputingServiceContainer* and handled by the *ComputingCAS*. The *ComputingServiceContainer* reifies the *VRComputingNode VirtualizedResource* that virtualizes a simple *ComputingHost* composed of the *ComputingResource_1* and the *StorageResource_1*.

The *AGIO* provides an excellent framework for modeling the structure of multi-agent-based grid systems. It contains a very extensive set of concepts describing relationships between agents operating within the grid as well as an interesting approach to enriching grid services with agents' intelligence through the use of Cognitive Environments. Unfortunately, it does not contain appropriate classes related to features such as physical resource description, resource discovery and brokering, or resource software requirements and/or capabilities. Therefore, while possibly a good candidate for future work on modeling the "universe" of *AiG* system's agents, it is not appropriate for use as a resource description ontology. Furthermore, it has to be noted that the *AGIO* project was actually an MS Thesis and currently is not pursued further.

7.5 Concluding remarks

The aim of this chapter was to summarize existing efforts at creating an ontology of the Grid, and underlying agent-grid integration. At the same time, the presented material was viewed from the perspective of the *Agents in Grid* project, which we are working on. Our main question was: is there an ontology of the Grid that we could adopt in our project?. Looking at the above list of attempts presented to directly define grid ontology or the use of reverse engineering to extract one, we believe we should reuse and extend the core grid ontology, which turns out to be the closest to our needs. Extensions will be based first, on other efforts (or experiences drawn from them); the *GLUE* project in particular, and the *UniGrid*. Second, we will have to introduce concepts that are specific to our effort, e.g. economic concepts related to contract negotiations and trust related concepts. We will report on our progress in subsequent publications.

Acknowledgments

Work of the Polish team was in part 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.

PROOF

References

- [1] I. Foster, N.R. Jennings, C. Kesselman, “Brain Meets Brawn: Why Grid and Agents Need Each Other”, Conference on Autonomous Agents and Multiagent Systems, International Joint, 1, 8–15, 2004.
- [2] F. Kordon, “Personal Communication”.
- [3] J. Hendler, “Agents and the Semantic Web”, *IEEE Intelligent Systems*, 16(2), 30–37, 2001.
- [4] N. Spivak, “Making Sense of the Semantic Web”, <http://www.slideshare.net/syawal/nova-spivack-semantic-web-talk>.
- [5] M. Drozdowicz, M. Ganzha, W. Kuranowski, M. Paprzycki, I. Alshabani, R. Olejnik, M. Taifour, M. Senobari, I. Lirkov, “Software Agents in ADAJ: Load Balancing in a Distributed Environment”, in M. Todorov, (Editor), “Applications of Mathematics in Engineering and Economics’34”, Volume 1067, *AIP Conf. Proc.*, 527–540, American Institute of Physics, College Park, MD, 2008.
- [6] M. Ganzha, M. Paprzycki, I. Lirkov, “Trust Management in an Agent-based Grid Resource Brokering System—Preliminary Considerations”, in M. Todorov, (Editor), “Applications of Mathematics in Engineering and Economics’33”, Volume 946, *AIP Conf. Proc.*, 35–46, American Institute of Physics, College Park, MD, 2007.
- [7] W. Kuranowski, M. Ganzha, M. Gawinecki, M. Paprzycki, I. Lirkov, S. Margenov, “Forming and managing agent teams acting as resource brokers in the Grid—preliminary considerations”, *International Journal of Computational Intelligence Research*, 4(1), 9–16, 2008.
- [8] M. Dominiak, M. Ganzha, M. Paprzycki, “Selecting grid-agent-team to execute user-job—initial solution”, *Proceedings of the Conference on Complex, Intelligent and Software Intensive Systems*, 249–256, IEEE CS Press, Los Alamitos, CA, 2007.
- [9] “Agents in the Grid Project”, <http://sourceforge.net/projects/gridagents>.
- [10] W. Kuranowski, M. Paprzycki, M. Ganzha, M. Gawinecki, I. Lirkov, S. Margenov, “Agents as resource brokers in grids—forming agent teams”, Volume 4818, 472–480, Springer, Berlin, 2007.
- [11] A. Chien, H. Casanova, Y.S. Kee, R. Huang, “The Virtual Grid Description Language: vgDL”, UCSD Technical Report CS2005-0817, University of California San Diego, 2005, http://www-csag.ucsd.edu/papers/VirtualGrids8-9-2004_V0.95TR.pdf.
- [12] Y.S. Kee, D. Logothetis, R. Huang, H. Casanova, A.A. Chien, “Efficient resource description and high quality selection for virtual grids”, *CCGRID ’05: Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid’05)*, Volume 1, 598–606, IEEE Computer Society, Washington, DC, USA, 2005.

- [13] “VGrADS—Virtual Grid Application Development Software”, <http://vgrads.rice.edu>.
- [14] J. Basney, M. Livny, “Deploying a High Throughput Computing Cluster”, in R. Buyya, (Editor), “High Performance Cluster Computing: Architectures and Systems”, Volume 1, Prentice Hall PTR, 1999.
- [15] T. Tannenbaum, D. Wright, K. Miller, M. Livny, “Condor – A Distributed Job Scheduler”, in T. Sterling, (Editor), “Beowulf Cluster Computing with Linux”, MIT Press, October 2001.
- [16] D. Thain, T. Tannenbaum, M. Livny, “Condor and the Grid”, in F. Berman, G. Fox, T. Hey, (Editors), “Grid Computing: Making the Global Infrastructure a Reality”, John Wiley & Sons Inc., December 2002.
- [17] D. Thain, T. Tannenbaum, M. Livny, “Distributed computing in practice: the Condor experience”, *Concurrency—Practice and Experience*, 17(2–4), 323–356, 2005.
- [18] R. Raman, M. Livny, M. Solomon, “Matchmaking: Distributed Resource Management for High Throughput Computing”, Proc. of the 7th IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July 1998.
- [19] R. Raman, M. Livny, M. Solomon, “Resource Management through Multilateral Matchmaking”, Proc. of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9), 290–291, Pittsburgh, PA, August 2000.
- [20] “Condor Manual”, <http://www.cs.wisc.edu/condor/manual/>.
- [21] D.W. Erwin, D.F. Snelling, “UNICORE: A Grid computing environment”, in R. Sakellariou, J. Keane, J. Gurd, L. Freeman, (Editors), “Euro-Par 2001 Parallel Processing”, Volume 2150, Lecture Notes in Computer Science, 825–834, Springer-Verlag, 2001.
- [22] I.T. Foster, “Globus Toolkit Version 4: Software for Service-Oriented Systems.”, in H. Jin, D.A. Reed, W. Jiang, (Editors), “NPC”, Volume 3779, Lecture Notes in Computer Science, 2–13, Springer, 2005.
- [23] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, “Grid Information Services for Distributed Resource Sharing”, 2001, <http://citeseer.ist.psu.edu/czajkowski01grid.html>—.
- [24] “MDS 2.4 Schemas”, <http://globus.org/toolkit/docs/2.4/mds/Schema.html>.
- [25] “GLUE Working Group Report”, <http://forge.gridforum.org/sf/projects/glue-wg>.
- [26] “GLUE Specification v. 2.0”, <http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.glue-wg/docman.root.public.comment/doc15227>, May 2008.
- [27] S. Andrezzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, J. Navarro, “GLUE v. 2.0—Reference Realizations to Concrete Data Models”, <http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.glue-wg/docman.root.public.comment/doc15219>, May 2008.

- [28] J. Heflin, "OWL Web Ontology Language use cases and requirements. W3C recommendation", Technical report, World Wide Web Consortium, 2004.
- [29] D.L. McGuinness, F. van Harmelen, "OWL Web Ontology Language overview. W3C recommendation", Technical report, World Wide Web Consortium, 2004.
- [30] H. Tangmunarunkit, S. Decker, C. Kesselman, "Ontology-based Resource Matching in the Grid—The Grid meets the Semantic Web", Proc. of the Second International Semantic Web Conference, Sanibel-Captiva Islands, 2003.
- [31] A.M. Pernas, M.A.R. Dantas, "Using Ontology for Description of Grid Resources", Proc. of the 19th International Symposium on High Performance Computing Systems and Applications, 223–229, IEEE Computer Society, Washington, DC, USA, 2005.
- [32] A. Vidal, F. da Silva e Silva, S. Kofuji, F. Kon, "Semantics-based grid resource management", Proc. of the 5th international workshop on Middleware for grid computing, 1–6, ACM, New York, NY, USA, 2007.
- [33] "Grid Ontologies: GO", <http://www.deinf.ufma.br/~vidal/>.
- [34] J. Brooke, D. Fellows, K. Garwood, C. Goble, "Semantic matching of grid resource descriptions", Proc. of the European Across Grids Conference 2004, 240–249, Springer, 2004.
- [35] "PCPack", <http://www.epistemics.co.uk/Notes/55-0-0.htm>.
- [36] M. Parkin, S. van den Burghe, O. Corcho, D. Snelling, J. Brooke, "The Knowledge of the Grid: A Grid Ontology", <http://www.cyf-kr.edu.pl/cgw06/presentations/c1-3.pdf>, CGW2006.
- [37] W. Xing, M.D. Dikaiakos, R. Sakellariou, S. Orlando, D. Laforenza, "Design and Development of a Core Grid Ontology", Proc. of the CoreGRID Workshop "Integrated research in Grid Computing", 21–31, November 2005.
- [38] "Core Grid Ontology", <http://grid.ucy.ac.cy/grisen/cgo.owl>.
- [39] L. Chunlin, L. Layuan, "Integrate software agents and CORBA in computational grid", *Comput. Stand. Interfaces*, 25(4), 357–371, 2003.
- [40] L. Chunlin, L. Layuan, "Agent framework to support the computational grid", *Journal of Systems and Software*, 70(1-2), 177–187, 2004.
- [41] I. Join submission, GMD FOKUS, "Mobile Agent System Interoperability Facilities Specification", Available at <http://www.omg.org/docs/orbos/97-10-05.pdf>, November 1997.
- [42] C. Jonquet, P. Dugenie, S.A. Cerri, "AGIL Specifications", Technical report, LIRMM, CNRS and University Montpellier II, France, 2006.
- [43] C. Jonquet, P. Dugenie, S.A. Cerri, "Service-based integration of Grid and Multi-Agent Systems models", Technical report, LIRMM, CNRS and University Montpellier II, France, 2006.
- [44] F. Duvert, "An ontology of GRID and Multi-Agent Systems integration", Master's thesis, University Montpellier II, 2006.
- [45] M. OConnor, H. Knublauch, S. Tu, B. Grosz, M. Dean, W. Grosso, M. Musen, "Supporting Rule System Interoperability on the Semantic Web with SWRL", *The Semantic Web ISWC 2005*, 974–986, Springer Berlin / Heidelberg, 2005.