CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **21**:1725–1741
Published online 13 February 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1395

Applying semantics to grid middleware

A. C. T. Vidal[1,*,†], F. J. S. Silva[1], S. T. Kofuji[2] and F. Kon[2]

[1]*Universidade Federal do Maranhão, Campus do Bacanga, São Luis, Maranhão, CEP 65085-581, Brazil*
[2]*University of São Paulo, Brazil*

SUMMARY

Scheduling parallel and distributed applications efficiently onto grid environments is a difficult task and a great variety of scheduling heuristics has been developed aiming to address this issue. A successful grid resource allocation depends, among other things, on the quality of the available information about software artifacts and grid resources. In this article, we propose a semantic approach to integrate selection of equivalent resources and selection of equivalent software artifacts to improve the scheduling of resources suitable for a given set of application execution requirements. We also describe a prototype implementation of our approach based on the Integrade grid middleware and experimental results that illustrate its benefits. Copyright © 2009 John Wiley & Sons, Ltd.

Received 15 July 2008; Revised 11 September 2008; Accepted 17 October 2008

KEY WORDS: grid computing; semantic grids; ontologies; OWL

1. INTRODUCTION

Computational grids are dynamic and heterogeneous environments that should deal efficiently with the coordinated sharing of resources for the solution of computational problems. A central issue in grid environments is the appropriate match of grid resources with application requirements. Well-known grid resource management systems focus their approaches to match grid resources and application requirements based only on resource descriptions. Resource management engines

*Correspondence to: A. C. T. Vidal, Universidade Federal do Maranhão, Campus do Bacanga, São Luis, Maranhão, CEP 65085-581, Brazil.
†E-mail: vidal@deinf.ufma.br

Contract/grant sponsor: CNPq
Contract/grant sponsor: The Brazilian Research Council; contract/grant number: 55.0895/2007-8

get resource requests from applications and try to find a suitable resource or a set of suitable resources over which applications can run. In general, resource discovery mechanisms, perform an exact matching of application requirements with attribute-based resource descriptions. More flexible mechanisms, as shown in [1], allow asymmetrical matching of inexact or incomplete application requirements with ontology-based resource descriptions.

A successful grid resource allocation depends, among other things, on the quality of the available information about software artifacts and grid resources. Strategies for application scheduling and scheduling algorithms are based on descriptions of resource attributes and application requirements and preferences. So, the way how this information is captured, stored, organized, and made available plays a relevant role in resource matching activities. Each of these tasks is built around the available grid resource and application metadata. Grid metadata permeates and connects them. From our point of view, the grid metadata design can affect the execution of tasks in different and relevant ways. To face this problem, we propose a semantic approach based on ontologies, whose objective is the efficient distribution of applications throughout the grid.

Ontologies can improve the quality of information about grid software and resources and this can help to increase the efficiency of software and resource management on the grid. A useful grid ontology should comprise a taxonomy of grid concepts, properties, and axioms that can be reused in different contexts and by other ontologies. Such ontology, used in conjunction with inference tools, provides a flexible and powerful way of reasoning about grid elements.

In this article, we present a set of related extensible grid ontologies and describe how this semantic approach may be used on grid environments, highlighting resource matching for scheduling application execution. Our prototype implementation uses the Integrade grid middleware [2]. Integrade follows an opportunist approach, taking advantage of idle computational resources for executing computationally intensive parallel applications. We describe how our semantic approach could be integrated to Integrade to allow better management and effective reuse of software and grid resources.

We also performed experiments, comparing the results of the conventional resource matching approach with the semantic one. The results sustain our claim that semantics improve the reuse, sharing, and integration of software and computational resources on grid environments.

This article is organized as follows: Section 2 introduces relevant semantic grid concepts and technologies. Section 3 presents a set of grid ontologies and their relationship for composing a grid knowledge base. Section 4 explores the use of semantics on grid environments considering two usage scenarios. Section 5 describes a proposed architecture for providing semantic services for grid environments and an implemented prototype based on the Integrade grid middleware. Section 6 describes experiments performed for evaluating our semantic approach for resource matching in comparison with the Integrade conventional one, as well as a performance evaluation of queries and inference on the knowledge base. Section 7 describes relevant related works, comparing them with our approach, while Section 8 presents our conclusions and describes future directions of this work.

2. SEMANTIC GRID BACKGROUND AND TECHNOLOGIES

de Roure *et al.* [3] define semantic grid as ‘an extension of the current grid in which information and services are given a well-defined meaning, better enabling computers and people to work in cooperation'. Semantic grids, as Semantic Web technologies, enable integration of applications, data, resource, and users in specific domains on the grid and eases automation of middleware tasks, such as selecting grid resources for application execution based on their requirements [4]. Additionally, such technologies also enable the interoperability between grid systems, since they can provide mappings between different grid middleware infrastructures.

Ontology is defined by Swartout and Tate [5] as an explicit representation of domain concepts that provides a basic structure around which knowledge bases can be built. Ontologies, as pointed out by Chandrasekaran *et al.* [6], can improve the quality of information about grid software and resource. They enable the reuse of domain knowledge, the sharing of common understanding about domain concepts, explicit definition of domain assumptions, and interoperability enhancement among other important features, in different grid domains.

Technologies applied on semantic grids are closely related to the ones applied on the Semantic Web. They are evolved from using languages such as the *Resource Description Language* (RDF) [7] to more complete approaches, such as the *Web Ontology Language* (OWL). RDF is a W3C standard for knowledge representation on the Semantic Web that does not enable the definition of resource meaning. *RDF Schema* (RDFS) [8] is a semantic extension of RDF, which defines classes and properties that can be used to describe other classes, properties, and resources on specific domains. A more complete approach for describing ontologies is through the use of OWL [9]. OWL extends RDF [10] and is derived from DAML+OIL [11] to represent ontologies.

OWL embodies the so-called *Classical Paradigm*: a modeling paradigm based on characteristics appropriate to the open environment of the Semantic Web [12]. In this article, we explore cases of inferencing and querying a grid ontology based on OWL-DL, a sublanguage of OWL. Ontologies in OWL-DL can be processed by a reasoner, through inference, to meet different goals. Conclusions not explicitly presented in the knowledge base can be inferred from rules and axioms in this base, finding what is necessarily true and consistent with the system axioms. Subsumption inference tasks produce a class hierarchy based on the class descriptions. It enables one to construct an asserted hierarchy and let a reasoner engine infer and maintain any additional inheritance relation. Such task is related to the domain definition knowledge and, in general, is useful to perform most subsumption inferences in advance, since they will apply to a domain as a whole. Additionally, performing this inference in advance enables us to submit queries over an inferred, i.e. updated ontology set. The reasoner can also perform consistency checking, and computation of inferred types, inferring whether or not it is possible for the class to have any instance, based on the description of a class [13].

3. GRID ONTOLOGIES

We used the Protégé-OWL tool [14] to enter the proposed set of grid ontologies and to work on refining it. This tool enables the creation of an extensible OWL ontology, including classes, properties, restrictions on classes, and individuals (instances). Additionally, reasoning to check consistency and to infer class hierarchy was performed through a connected Pellet [15] reasoning tool.

We defined our system architecture based on the grid system ontology described in [16], which includes a large and unordered variety of different, although related, concepts, properties, and axioms. That ontology was an exploratory approach to expose the taxonomic and reasoning possibilities on the grid domain. To improve the knowledge base maintenance and scalability, we separated the original ontology in a set of related ontologies of three types: upper-level ontologies, ancillary ontologies, and *concrete* ontologies.

A *upper-level ontology* or *top-level ontology*, as defined in [6], is an 'ontology that describes knowledge at higher levels of generality'. In general, upper-level ontologies are used to describe basic concepts that can be extended in more specific ontologies. Ancillary ontologies contain domain-specific concepts. We consider as ancillary an ontology that supports another one, but performing a less important role. Concrete grid ontologies extend the upper-level ontologies to describe specific grid concepts, reusing the top ontologies. These different ontologies can be connected through an *import* mechanism.

The *import* mechanism allows an ontology to import other ontology. This mechanism permits referring and extending all concepts and properties of the imported ontology, enhancing knowledge reuse and sharing. Concepts and properties from the imported ontology receive a prefix referring to the related name space. Other known mechanisms to relate ontologies, not explored in this work, are the so-called *mapping* and *e-connection* mechanisms. These approaches are relevant to improve the knowledge base scalability. In our knowledge base prototype, we have one upper-level ontology, which is called the *Grid Base Ontology*, one ancillary ontology, the *Platform Ontology*, and a concrete grid ontology, the *Grid Resource Management Ontology*. The acronyms *po* and *gbo* are, respectively, namespaces of the *Platform Ontology* and the *Grid Base Ontology*, imported by the *Grid Resource Management Ontology*.

Figure 1 shows the asserted class hierarchy of a specific *Platform Ontology* branch, the operating system branch. The *Platform Ontology* contains a short class hierarchy and related properties about the computational platform domain. It includes, among others, concepts such as operating systems,

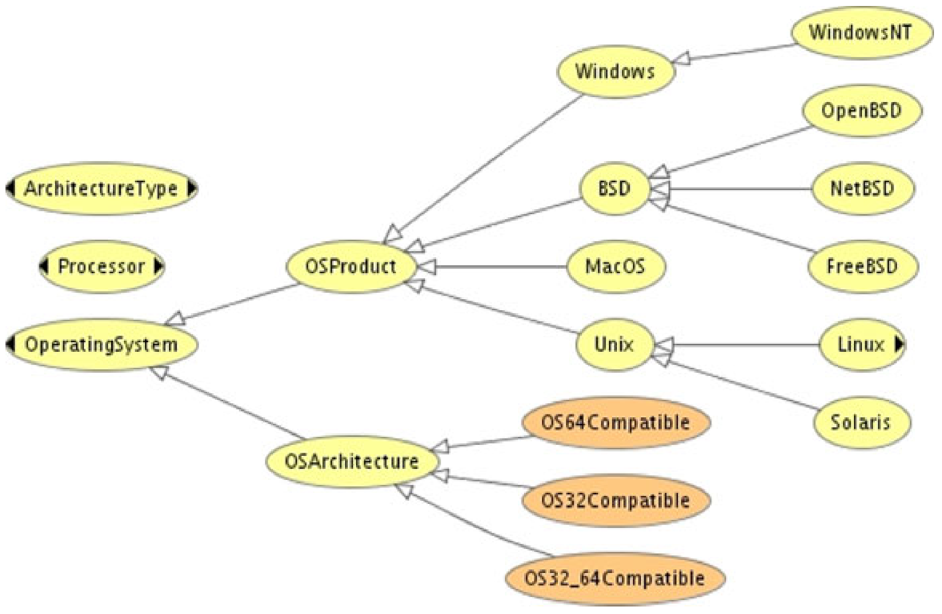

Figure 1. Platform ontology—OS branch.

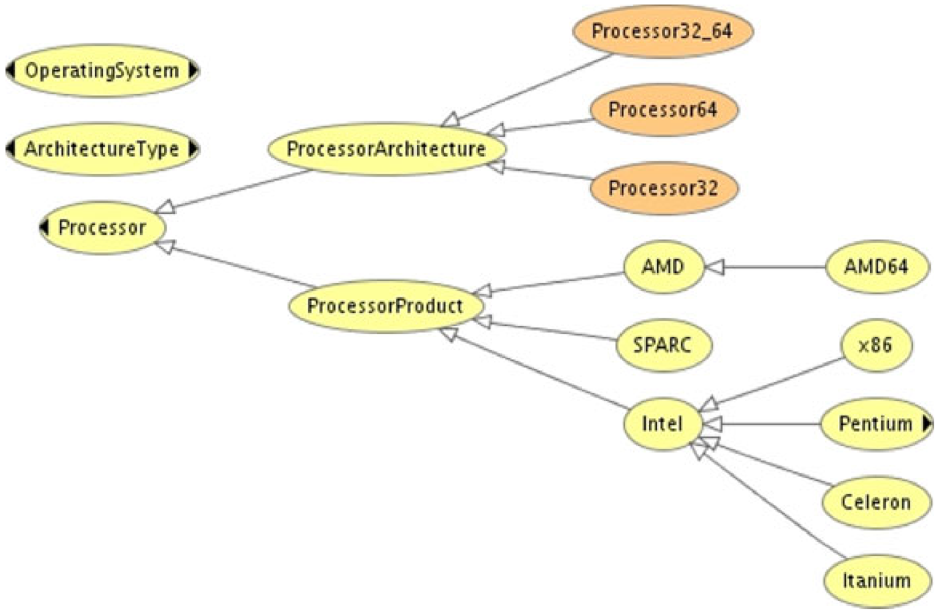

Figure 2. Platform ontology—processors.

architectures, and processors. This hierarchy can be extended with new concrete concepts, allowing to obtain new knowledge from incomplete asserted information and based on axioms defined in the ontology. Figure 2 presents the asserted class hierarchy of another specific *Platform Ontology* branch, the processor branch.

According to Rector *et al.* [17], *Defined classes* are classes for which there is at least one set of necessary and sufficient conditions. We use OWL properties to state the necessary and sufficient conditions for class membership. For example, a class named *ComputerLinux64bits* could be *defined*, through the property *hasProcessor*, as related to the class *Processor64bits*, and through the property *hasOS* as related to the *Linux* class. Consider an individual named *Corel2Duo2.6GHz*, which is an instance of the *Processor64bits* class and an individual named *Ubuntu8.0*, which is an instance of the *Linux* class. An individual of the *Computer* class that is related to the *Corel2Duo2.6GHz* instance and to the *Ubuntu8.0* instance is inferred to be an instance of the *ComputerLinux64bits* class. This inferred classification can be used to identify grid machines belonging to the *ComputerLinux64bits* class even if the individual was not explicitly asserted as an instance of this class, allowing for the automatic discovery of suitable grid nodes for executing any application implemented for the Linux 64 bit platform.

The *Grid Base Ontology* initially acts as a fundamental taxonomy encompassing the main concepts related to grid systems. It contains concepts, properties, and axioms that can be considered common to grid domain applications and users. Figure 3, generated by the Protégé-OWL tool, shows the asserted class hierarchy of the *Grid Base Ontology*. The class hierarchy is based on two root concepts: *Grid Software Concepts* and *Grid Resource Concepts*. The former encompasses other

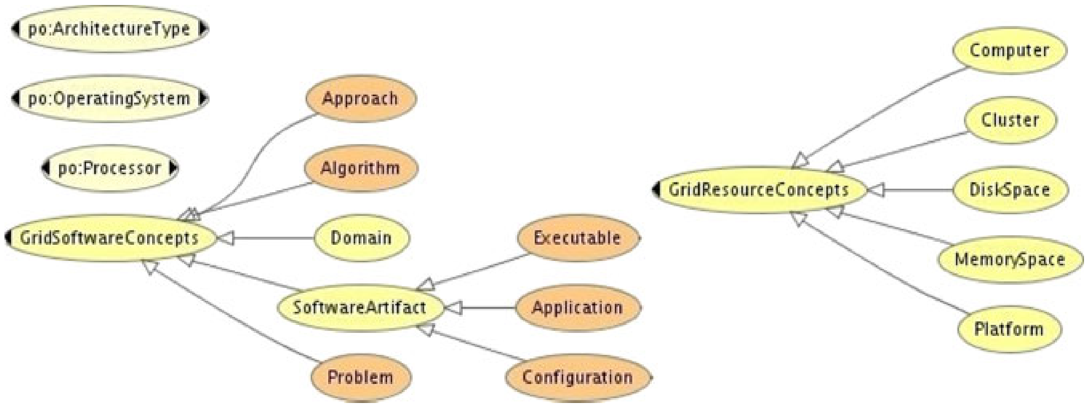

Figure 3. Grid base ontology.

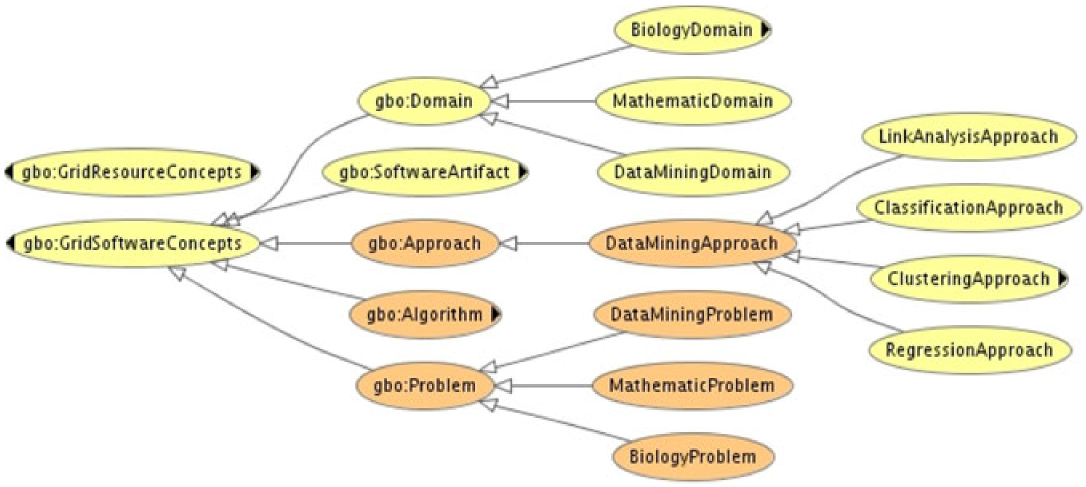

Figure 4. GRM ontology—software concepts.

related concepts, such as *Domain*, *Problem*, *Approach*, *Algorithm*, *Application*, *Software Artifacts*, and so on. These concepts are sufficiently high level to be extended by *Grid Application Developers* to describe more specifically their domains, problems, and related software solutions. *Grid Resource Concepts* encompasses concepts related to grid computational resources, such as *Computer*, *Cluster*, *DiskSpace*, *MemorySpace*, and *Platform*. These concepts, in turn, can be extended by the *Grid Manager* to describe a concrete computational grid infrastructure. The upper-level ontology imports an ancillary ontology, the *Platform Ontology*, and reuses its concepts and properties.

An example of a more specific grid ontology is the *Grid Resource Management Ontology*. To improve visibility, we separated parts of this ontology into two distinct figures. A set of more

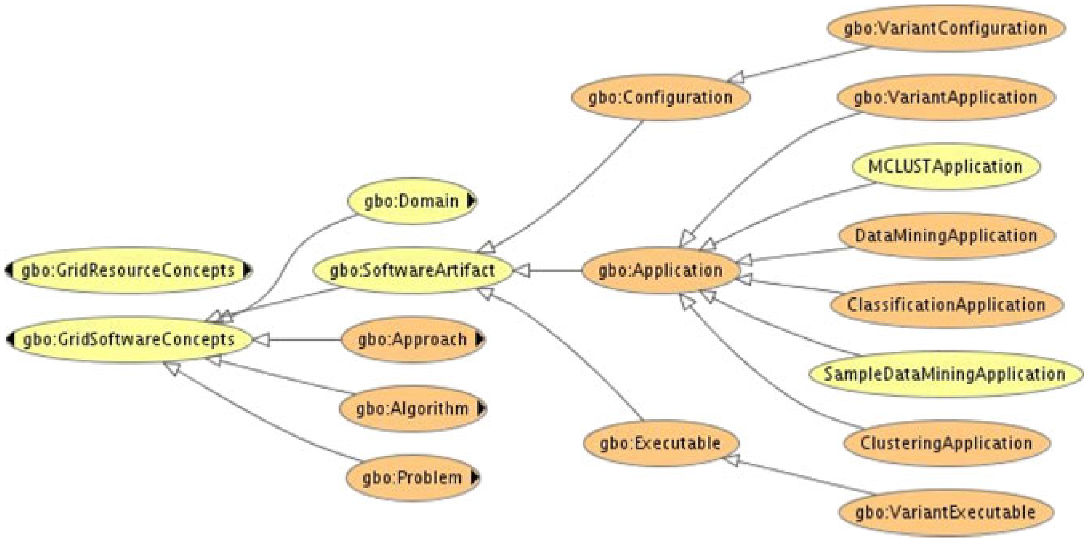

Figure 5. GRM ontology—software artifacts.

general software related concepts is presented in Figure 4, while some concepts related to software artifacts are shown in Figure 5. This ontology imports the *Grid Base Ontology* showed in Figure 3. The set of described concepts includes a specific domain, *Data Mining Domain*, and some of its specific related concepts, also derived from the *Grid Base Ontology*, such as *Data Mining Problems*, *Classification Approaches*, and so on. Other different domains and their related concepts can be created as extensions of the *Grid Base Ontology*, such as *Mathematic* and *Biology*. The complete set of ontologies is available for download at http://www.deinf.ufma.br/∼vidal/gbos.html.

4. GRID ONTOLOGIES CASE STUDY

This section describes two scenarios of exploring the ontologies to improve resource management on grid environments. Initially, we discuss how grid scheduling can take advantage of a semantic resource pre-selection. Afterwards, we describe how the proposed ontologies can leverage the grid usage as a distributed and integrated application repository.

For both the scenarios, we adopted an *inference in advance* policy, for the whole set of involved ontologies. A subsumption inference updates the knowledge base in a suitable frequency. The inferred knowledge base contains new inheritance relationships, extending the available metadata with new semantic information.

4.1. Semantic grid scheduling

According to [18], a schedule is a mapping of tasks to time and computational resources and the *scheduling problem* consists in computing a schedule that optimizes a certain metric. Grid resource and application metadata play a crucial role in such scheduling task, since they describe resourcex features and application requirements in the grid context.

Grids are heterogeneous environments. In the same way, grid metadata themselves are also heterogeneous, making it difficult to explore grid resources efficiently. For example, different virtual organizations may employ distinct terminologies to describe the same grid resource. As a result, the integration of related grid resources is not an easy task. A semantic approach can mitigate this problem enabling the identification of related resources based on ontologies concepts and axioms.

In general, whenever a grid user requests an application execution, the grid middleware schedules a set of resources for performing the computation. A semantic approach can also achieve better resource matching than an approach based on exact matching of metadata attributes, leading to more alternative nodes for scheduling the application. This is due to the fact that, through an inference policy, which extends the set of selected resources, the grid middleware can discover interchangeable or equivalent resources, considering the user restrictions for a given application submission (such as the required minimum memory and processing power).

4.2. Semantic application repository

Computer grids have been used to solve problems in several areas of scientific, enterprise, and industrial activities, such as computational biology, image processing for medical diagnosis, weather forecast, high energy physics, marketing simulations, and oil prospection. Grid computing empowers the conception of a new generation of applications that allow combining computations, experiments, observations, and data collected in real time. The phenomena modeled by these applications require diverse software components whose compositions and interactions are extremely dynamic. Under particular conditions, application executables in a domain can be interchangeable and software components can be reused in different domains. We propose to leverage the grid usage as a distributed and integrated repository of software artifacts through the use of metadata.

To be able to select the best application component for each situation, it is important to have the means to classify these software artifacts and be capable of explicitly reasoning about them. The search mechanism for a software artifact should also be flexible allowing to obtain equivalent alternatives for a proposed query.

Using our proposed ontologies, each virtual organization could feed the knowledge base during the application registering process, either extending the hierarchy of software concepts or creating individuals ones to represent the software component. Later on, the knowledge base could be enhanced through a subsumption inference. Bulk policies can be adopted, to reduce the reasoner cost. Our semantic data structure privileges the class hierarchy and the subsumption inference, since both abstractions are central in ontology-based approaches.

While our approach focuses on enhancing application selection for executing a given user task, future developments could embody advances from the automated semantic software composition field [19] that encompass additional challenges such as the definition of extensible ontologies to define software components, their interfaces, granularities, requirements, restrictions, and policies for interchangeable use. Those extensions could be used in areas of related domain such as grid workflow composition and grid and grid services discovery and composition [20].

5. INTEGRADE KNOWLEDGE ARCHITECTURE

This work was developed in the context of the Integrade project[‡]. [2], a multi-university effort to build a novel grid computing middleware infrastructure to leverage the idle computing power of personal workstations for the execution of computationally intensive parallel applications.

We defined a general architecture for enabling the grid middleware and its users to take advantage of semantic services, improving grid utilization. While we developed a prototype based on the InteGrade middleware, our knowledge architecture is independent of the grid platform, allowing it to be reused on other grid projects.

This section presents a brief overview of the InteGrade grid middleware, describes the proposed grid knowledge architecture, the implementation of the architecture components, and the required InteGrade extensions for using the supplied semantic services.

5.1. InteGrade architecture overview

The basic architectural unit in an InteGrade grid is the cluster, a collection of machines usually connected by a local network. Clusters can be organized in a hierarchical federation, allowing the aggregation of a large number of machines. Each cluster contains a *Cluster Manager* node that executes InteGrade components responsible for managing the cluster computing resources and for inter-cluster communication. Other cluster nodes are called *Workstations*, which export part of its resources to Grid users. They can be shared or dedicated machines.

InteGrade currently allows the execution of three application classes:

1. Regular sequential applications: The application executable code is assigned to a single grid node.
2. Parametric or bag-of-task (BoT) applications: Several copies of the application executable code is assigned to different grid nodes. Each node receives a subset of the application input data and computes the results in parallel with the other ones. In this case, there is no communication between the nodes.
3. Parallel applications that follow the bulk synchronous parallel (BSP) or message passing interface (MPI) models: several nodes are assigned for application execution. Computations take place on every participating node and processes exchange data between themselves.

The InteGrade architecture comprises several components. The most fundamental ones, which enable application execution, are:

- *Application Repository* (AR): Before being executed, an application must be previously registered with the Application Repository. This component stores the application description (metadata) and executable code.
- *Application Submission and Control Tool* (ASCT): A graphical user interface that allows users to browse the content of the Application Repository, submit applications, and control their execution. It is available both as a Web portal and as a stand-alone application.

[‡]Homepage: www.integrade.org.br

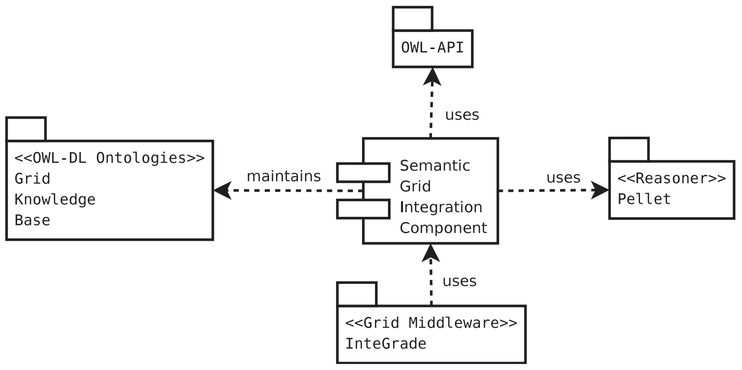

Figure 6. Semantic grid integration architecture.

- *Local Resource Manager* (LRM): A component that runs in each cluster node, collecting information about the state of resources such as memory, CPU, disk, and network. It is also responsible for instantiating and executing applications scheduled to the node.
- *Global Resource Manager* (GRM): Manages cluster resources by receiving notifications of resource usage from the cluster LRMs (through an information update protocol) and runs the scheduler that allocates tasks to nodes based on resource availability; it is responsible for communication with GRMs in other clusters for inter-cluster application execution.
- *Execution Manager* (EM): Maintains information about each application submission, such as its state, executing node(s), input and output parameters, submission, and termination timestamps. It also coordinates the recovery process in case of application failures.

5.2. InteGrade knowledge architecture goals

We defined a general architecture for enabling the grid to take advantage of the semantic approach. The main purpose of such architecture is to improve grid utilization. The defined architecture is based on the following requirements:

1. The architecture must be independent of the grid platform. While we developed a prototype based on the InteGrade middleware, the architecture components can be reused on other grid software infrastructures, since they provide semantic-based services consumed by grid components, such as its scheduler (for providing a better resource matching mechanism) or the grid application submission tool, which allows the user to browse the knowledge base for application discovering.
2. It must provide the means to create and maintain an extensible grid knowledge base. Knowledge base extension involves the extension of the concepts asserted in the upper-level ontologies and instance creation of specific classes in concrete ontologies.
3. The reasoning mechanisms and policies must infer knowledge from incomplete asserted information.
4. It must provide the means to enable grid users to perform queries on the knowledge base, enhancing application, and grid resource discovery.

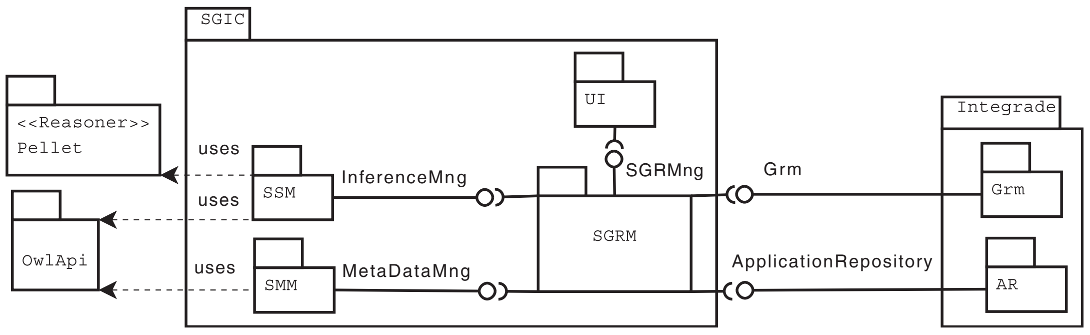

Figure 7. SGIC internal components.

5.3. Proposed Grid Knowledge Architecture

We developed a prototype of a *Semantic Grid Integration Architecture* depicted in Figure 6. Its main component is the SGIC *Semantic Grid Integration Component*, which is a facade module that uses other more specific components to provide its service. SGIC uses the OWL-API and implementation [21] for manipulating the content and structure of the knowledge base, performing tasks such as creating classes and instances, and executing queries. It also uses the Pellet reasoner for performing inference tasks, such as classifying registered applications based on the Grid Management class hierarchy. An SGIC internal component encapsulates all interactions between the proposed Semantic Grid Architecture and the underlying grid infrastructure.

Figure 7 shows SGIC internal components (SGRM, SMM, SSM, and UI) and their connection with InteGrade components, the AR and the global resource manager (GRM). The semantic grid resource manager (SGRM) mediates interactions between grid users, the integrade middleware, and others SGIC components. The SGRM uses the semantic metadata manager (SMM) for maintaining the semantic metadata repository comprising grid ontologies, classes, and instances and the semantic services manager (SSM) for performing inference tasks, such as concept hierarchy classification, ontologies consistency check, and computation of inferred types. Both SSM and SMM use the OWL-API, while the SSM also uses the *Pellet* reasoner.

A grid user registers an application on the grid through an User Interface (UI) that calls the SGRM component. The latter uses the *SMM* to store metadata about the registered application on the knowledge base and uses the InteGrade AR for storing the application executable files. Application metadata include the application name, which algorithm it implements, which platform it runs on, and its type (Regular, BoT, MPI, or BSP). The system then uses inference mechanisms to relate the application metadata to the existing software concepts hierarchy and their individuals.

The *SSM* component infers new class hierarchies and types of registered instances. The inferred data are used as a base for subsequent queries submitted by grid users. For example, the inferred metadata is used to assist the user on selecting the appropriate application for performing a desired task. Given an application submission, inferred data concerning interchangeable binaries and grid nodes architecture are used for enlarging the set of grid nodes capable of executing the requested application.

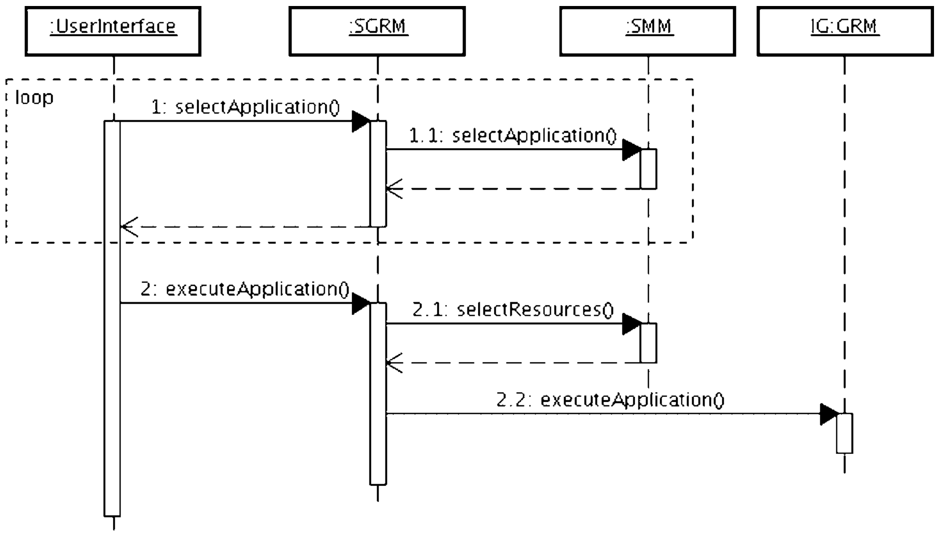

Figure 8. Sequence diagram—Selecting resources before application execution.

The sequence diagram in Figure 8 shows how one can use the semantic grid enhancements for selecting an application for executing a desired task and for submitting an execution request to the grid. Via the UI component one initiates an interactive loop through the *SGRM*:*selectApplication*() method which, in its turn, calls *SMM*:*selectApplication*(). The interaction loop follows a refinement process until an specific application that matches the user requirements (desired task) is selected. The user then submits an execution request, calling the *SGRM*:*executeApplication*() method. The SGRM calls the *SMM*:*selectResources*() method that uses the inferred knowledge base for discovering the grid nodes capable of executing the requested application. The resource set is then forwarded to the grid scheduler (the GRM component on InteGrade) for performing the appropriate resource mapping.

While the described prototype was based on the InteGrade grid middleware, our Semantic Grid Integration Architecture was designed to be sufficiently general to be ported to other grid software infrastructures with minimum effort. This requirement is supported mainly by the SGRM component, since it encapsulates all interactions between the proposed semantic architecture and the underlying grid middleware. It also only requires concrete implementations of two abstractions commonly available on grid environments: an application repository and the scheduler. The interactions with these two grid components are provided through two abstract classes that must be specialized for the specific grid middleware being used.

6. MODEL EVALUATION

To evaluate our approach and demonstrate its usefulness, we performed experiments in a scenario of resource scheduling for application submissions, comparing the result of the conventional InteGrade resource selection with the one obtained with our semantic enhancements.

6.1. Resource matching

The grid environment hardware considered on the experiments is described on Table I. Platforms P1, P2, P3, P6, and P7 execute the *i686 GNU/Linux* operating system, platforms P4 and P5 the *amd_64 GNU/Linux*, and platform P8 the *PowerPC_Darwin* operating system. From the application execution perspective, platforms P1, P2, P3, P6, and P7 are equivalents and they correspond to 19 grid nodes. Platforms P4 and P5 are also equivalent and the grid environment has one node of each type, comprising two grid nodes. Only one grid node follows platform P8, which has no other equivalent platforms.

For performing the experiments, we used several software artifacts consisting of a set of algorithms for *Image Texture Analysis* (ITA) related to the *Pattern Recognition* domain. The algorithms describe conventional texture analysis methods, e.g., the Gray-Level Run-Length Method (GLRLM) and the Spatial Gray Level Dependency Method (SGLDM), used to extract characteristics from a given image. The algorithms can be used in different horizontal domains, such as aerial photo image analysis for geographic research and tomography image analysis for cancer diagnosis.

The methods used by the different algorithms lead to different CPU and memory requirements but, under a specific context, these algorithms could replace each other. Table II shows the used set of algorithms and applications and the suitable platforms for executing each one of them, considering the available executable binaries.

On InteGrade, during the submission process, the user can either select a specific application executable binary or an application, in a more general sense, to be executed by the grid. In the first case, the middleware will try to identify the nodes on the local cluster whose platforms are described exactly as the platform description of the correspondent binary. The second case is more flexible,

Table I. Hardware platform types.

Type	Platforms
P1	Intel(R) Pentium(R) 4 CPU 3.00 GHz
P2	AMD Athlon(tm) XP 2800+
P3	Intel(R) Xeon(TM) CPU 3.00 GHz
P4	AMD Athlon(tm) 64 Processor 3200+
P5	AMD Athlon(tm) 64 Processor 3000+
P6	Intel(R) Pentium(R) 4 CPU 2.80GHz
P7	Intel(R) Pentium(R) D CPU 2.80GHz
P8	Power PC G4

Table II. ITA available software artifacts.

Application	Platforms	#Nodes
Suitable platforms and nodes		
SGLDM_App	P1	8
GLRLM_App	P5 and P8	2
GLDM_App	P4	1
PSM_App	P4	1

Table III. Comparison of scheduling approach results.

Application	#Selected nodes by approach		
		Semantic	
	InteGrade	First case	Second case
SGLDM_App	8	19	22
GLRLM_App	2	3	22
GLDM_App	1	2	22
PSM_App	1	2	22

in the sense that the middleware will try to first discover all available executables of the selected application. Such an approach increases the set of suitable platforms to execute the application and, potentially, will increase the number of available machines for application execution.

The experiment consists of submitting a user request for each ITA application (SGLDM, GLRM, GLDM, and PSM) for the conventional InteGrade middleware and for our semantics-based extended version, comparing the resource matching results obtained by both approaches. For the conventional InteGrade, we used the more general case of submitting an application, since it increases the set of suitable platforms to execute the application, as explained previously.

Table III shows the results obtained. The conventional InteGrade discovered 8, 2, 1, and 1 nodes available for executing applications SGLDM, GLRM, GLDM, and PSM, respectively, considering the grid hardware described in Table I. The second column of Table III (first case) shows the results obtained with our semantic approach using the equivalence relationships between platforms asserted with the *Platform Ontology*. The third column of Table III (second case) shows the results obtained when our semantic approach also considers the use of equivalent applications related to a set of equivalent algorithms, defined on the *Grid Management Ontology*. The better results are due to the fact that algorithms described through the assertion *gbo:describes some TextureAnalysisApproach* are inferred to be also subclass of the class *TextureAnalysisApproach*. This happens because the class *TextureAnalysisApproach* is a subclass of the class *Algorithm* and is also *defined* by the assertion *gbo:describes only TextureAnalysisApproach*.

As the experimental results demonstrated, the use of semantics can increase the amount of possibilities for executing a given user application request, since each application can be related to different but equivalent executables. In the same way, each executable can be related to different but equivalent platforms, leading to better resource matching results.

6.2. Query and inference performance analysis

We executed two experiments with the goal of evaluating the cost of performing queries and inferences over the proposed grid ontology. In the first experiment, 100 different SPARQL[22] queries were built around applications and computing resources selection on the grid.

The generated queries were executed through an application based on the OWL-API. We executed each query on a database with different amounts of individuals (1000, 5000, and 8000 individuals), measuring the query execution time. Table IV presents the maximum, minimum, and average execution time of the 100 queries for each database size. The experiment was executed on an Intel

Table IV. Query execution time in ms.

#Instances	$T_{\max}$	$T_{\min}$	Average	Standard deviation
Queries on the platform ontology				
1000	2.39	0.35	0.55	0.39
5000	3.15	0.34	0.57	0.50
8000	3.48	0.36	0.59	0.48

Table V. Class inference execution time in seconds.

#Instances	Time
Inference on platform ontologies	
1000	4.64
5000	4.77
8000	4.94

Pentium M machine with 1.70 GHz, 496 MB of RAM, running the Ubuntu 7.04 (feisty) Linux distribution.

The average execution time spent on each query was a little more than 0.5 ms, which is an irrelevant overhead for a typical grid application that normally requires several hours or even days to be executed. The main reason for this small execution time is that the SPARQL queries did not need to perform inferences over the knowledge base, since they were realized over a previously inferred knowledge base.

We executed a second experiment that evaluated the inference mechanism cost. In this experiment, we executed the inference of class hierarchy for databases with the same amount of individuals as used on the query experiment (1000, 5000, and 8000 individuals). Table V shows the results obtained, in seconds.

The results highlight the need for a well-defined execution policy for the inference mechanism, since this task has a considerable computational cost, which makes it not practical to execute it for every new query on the knowledge base. Our approach is to execute the inference mechanism whenever the knowledge base is modified only at specified time intervals, defined by the grid administrator or when the grid is underloaded, an information obtained through the InteGrade Execution Manager component.

7. RELATED WORK

The core grid ontology (CGO), described in [23], provides a common knowledge base about grid systems. In this sense, our approach is similar to the one presented in this article. CGO is an extensible grid system ontology that expresses fundamental grid-specific concepts and relationships, without exploring inference mechanisms. However, our focus is on improving grid uses, such as application selection and execution, while exploring reasoning capabilities and query languages.

Cannataro [24] describes the Data Mining for Grid Programming project, which uses an ontology (DAMON) for the data mining domain. It describes how the ontology is used to enhance the design of distributed data mining applications. DAMON exploits only one specific grid application domain, namely, data mining, while we propose a more general approach for the management of grid content and resources, based on extensible ontologies for describing different application domains and grid resources.

Matching of grid resources and application requirements is largely explored by semantic grid applications. Brook *et al*. [25] propose a semantic approach to build a broker of resources described by different grid middlewares. They aim to provide seamless access to resources on the grid. Tangmunarunkit *et al.* [26] introduce a similar approach. In a similar way, Siddiqui *et al.* [1] describes an asymmetric mapping between applications and resources based on incomplete application requirement description. All of them propose an ontology-based matching of task requirements and grid resource policies. Our work, in contrast, aims to integrate selection of equivalent resources and selection of equivalent software artifacts, enhancing the scheduling of resources suitable for a given set of applications.

In our application and resource matching perspective, we borrow the general idea of *decoupled scheduling* from [18]. However, while they try to reduce the amount of selected resources, aiming to optimize a scheduling algorithm, our work aims to identify the maximum amount of suitable resources for satisfying a user request on a grid.

8. CONCLUSIONS

This article explores the use of ontologies to describe core concepts related to grid applications and resources. Such ontologies are composed with an *import* mechanism and can be extended either by adding new classes and individuals to a specific ontology or by connecting new ontologies derived from the original top-level ontologies. We described some important scenarios for exploring the knowledge base that can be constructed through the use of the proposed ontologies and the inference mechanisms that can improve the performance of grid scheduling and leverage the grid usage as a distributed and integrated application repository, leading to better discovery and composition of software artifacts. We propose a *Semantic Grid Integration Architecture* based on the OWL-API and the Pellet reasoner to be used for grid knowledge base maintenance and to explore the described scenarios. A prototype was developed using the InteGrade grid middleware as its base.

We also evaluated our approach, comparing the resource matching results for application submissions obtained with and without the use of semantics. This evaluation led to the conclusion that the use of semantics can bring substantial gains in resource management on grid environments and that, if proper care is taken, the performance overhead is negligible.

Future directions of this work include the investigation of the semantic relations between ontologies that have intersection, mapping [27], and between disjoint ontologies, e-connections [28], which comprehends a relevant aspect to improve the scalability of the knowledge base. Another interesting work would be to implement the proposed architecture on a different grid middleware, evaluating how much of the implementation source code could be effectively reused.

REFERENCES

1. Siddiqui M, Fahringer T, Hofer J, Toma I. Grid resource ontologies and asymmetric resource-correlation. *Second International Conference on Grid Service Engineering and Management* (*GSEM'05*) (*Lecture Notes in Informatics*), German Society of Informatics (eds.). Erfurt, Germany, 19–22 September 2005.
2. Goldchleger A, Kon F, Goldman A, Finger M, Bezerra GC. InteGrade: Object-oriented grid middleware leveraging idle computing power of desktop machines. *Concurrency and Computation*: *Practice and Experience* 2004; **16**:449–459.
3. de Roure D, Jennings NR, Shadbolt NR. The semantic grid: Present, past, and future. *Proceedings of IEEE* 2005; **93**(3):669–681.
4. Roure DD. A brief history of the semantic grid. *Semantic Grid*: *The Convergence of Technologies* (*Dagstuhl Seminar Proceedings*, vol. 05271), Goble C, Kesselman C, Sure Y (eds.). IBFI: Schloss Dagstuhl, Germany, 2005.
5. Swartout W, Tate A. Guest editors' introduction: Ontologies. *IEEE Intelligent Systems* 1999; **14**(1):18–19.
6. Chandrasekaran B, Josephson J, Benjamins V. What are ontologies, and why do we need them? *Intelligent Systems and their Applications*, *IEEE* 1999; **14**(1):20–26.
7. W3C. Resource Description Framework (RDF). http://www.w3.org/RDF/, April 2007.
8. W3C. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema, April 2007.
9. W3C. Web Ontology Language (OWL). http://www.w3.org/2004/OWL, April 2007.
10. Manola F, Miller E. RDF-PRIMER, W3C Recommendation, February 2004.
11. Pagels M. The DARPA Agent Markup Language Homepage, 2005.
12. Patel-Schneider PF, Horrocks I. Position paper: A comparison of two modelling paradigms in the semantic Web. *WWW '06*, Edinburgh, Scotland, May 2006, ACM Press: New York, NY, 2006.
13. Horridge M. A practical guide to building OWL ontologies with the Protégé-OWL plugin. Available at: http//www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf, 2004.
14. Knublauch H, Fergerson RW, Noy NF, Musen MA. The protégé owl plugin: An open development environment for semantic web applications. *ISWC2004*, Hiroshima, Japan, 2004.
15. Sirin E, Parsia B, Grau BC, Kalyanpur A, Katz Y. Pellet: A Practical owl-dl reasoner. *Technical Report 2005-68*, Maryland, U.S.A., 2005.
16. Vidal ACT, Jose de RP, Braga J, Kon F, Kofuji ST. Defining and exploring a grid system ontology. *MCG '06*: *Proceedings of the 4th International Workshop on Middleware for Grid Computing*, Melbourne, Australia. ACM Press: New York, NY, U.S.A., 2006; 16.
17. Rector A, Drumond N, Horridge M, Rogers J, Knublauch H, Stevens R, Wang H, Wroe C. OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. *Engineering Knowledge in the Age of the Semantic Web* (*Lecture Notes in Computer Science*). Springer: Berlin/Heidelberg, 2004; 63–81.
18. Zhang Y, Mandal A, Casanova H, Chien AA, Kee Y-S, Kennedy K, Koelbel C. Scalable grid application scheduling via decoupled resource selection and scheduling. *Sixth IEEE International Symposium on Cluster Computing and the Grid* (*CCGrid 2006*), Singapore, vol. 1, 2006; 568–575.
19. Fujii K, Suda T. Dynamic service composition using semantic information. *ICSOC'04*: *Proceedings of the 2nd International Conference on Service Oriented Computing*. ACM: New York, NY, U.S.A., 2004; 39–48.
20. Bubak M, Gubala T, Kapalka M, Malawski M, Rycerz K. Workflow composer and service registry for grid applications. *Future Generation Computer System* 2005; **21**(1):79–86.
21. Bechhofer S, Lord P, Volz R. The semantic web with the OWL API. *Proceedings of the 2nd International Semantic Web Conference* (*ISWC 2003*), Sanibel Island, Florida, U.S.A., April 2003.
22. W3C, SPARQL Query Language for RDF, May 2007. http://www.w3.org/TR/rdf-sparq1-query/.
23. Xing W, Dikaiakos MD, Sakellariou R. A core grid ontology for the semantic grid. *CCGrid 2006*, Singapore, May 2006. IEEE Computer Society: Silver Spring, MD, 2006; 178–184.
24. Cannataro M, Comito C. A data mining ontology for grid programming. *Workshop on Semantics in Peer-to-Peer and Grid Computing* (*in Conj. with WWW2003*), Budapest, Hungary, 2003; 113–134.
25. Brooke J, Fellows D, Garwood K, Goble C. Semantic matching of grid resource descriptions. *AxGrids 2004*, Nicosia, Cyprus (*Lecture Notes in Computer Science*, vol. 3165), January 2004; 240–249.
26. Tangmunarunkit H, Decker S, Kesselman C. *Ontology-Based Resource Matching in the Grid—The Grid Meets the Semantic Web* (*Lecture Notes in Computer Science*, vol. 2870). Springer: Berlin, 2003; 706–721.
27. Haase P, Motik B. A mapping system for the integration of OWL-DL ontologies. *IHIS '05*: *Proceedings of the First International Workshop on Interoperability of Heterogeneous Information Systems*, Breman, Germany. ACM: New York, NY, U.S.A., 2005; 9–16.
28. Grau BC, Parsia B, Sirin E. Combining OWL ontologies using E-Connections. *Web Semantics*: *Science*, *Services and Agents on the World Wide Web* 2006; **4**(1):40–59.