Available online at www.sciencedirect.com
SCIENCE DIRECT
Web Semantics: Science, Services and Agents on the World Wide Web 4 (2006) 1–13
JOURNAL OF Web Semantics
www.elsevier.com/locate/websem

Semantic approach to service discovery in a Grid environment

Simone A. Ludwig [a,*], S.M.S. Reyhani [b]

[a] *School of Computer Science, Cardiff University, Cardiff CF24 3AA, UK*
[b] *Department of Information Systems and Computing, Brunel University, Uxbridge UB8 3PH, UK*

Received 5 December 2003; received in revised form 26 November 2004; accepted 12 April 2005

Abstract

The fundamental problem that the Grid research and development community is seeking to solve is how to coordinate distributed resources amongst a dynamic set of individuals and organisations in order to solve a common collaborative goal. The problem arises through the heterogeneity, distribution and sharing of the resources in different virtual organisations. Interoperability is a main issue for applications to function with the Grid. This paper proposes a matchmaking framework for service discovery in Grid environments based on three selection stages which are context, semantic and registry selection. It provides a better service discovery process by using semantic descriptions stored in ontologies which specify both the Grid services and the application knowledge. The framework permits Grid applications to specify the criteria a service request is matched with and enables interoperability for the matchmaking process. A proof of concept is done with a prototype implementation, and an enhancement of the matchmaking process is achieved with a similarity metric which allows quantifying the quality of a match. A qualitative and quantitative evaluation of the prototype system is given with an analysis and performance measurements to quantify the scalability of the prototype.

Keywords: Interoperability; Ontology; Service discovery; Grid

1. Introduction

In mid 1990s Ian Foster and Carl Kesselman proposed a distributed computing infrastructure for advanced science and engineering which they called "The Grid". The vision behind the Grid is to supply computing and data resources over the Internet seamlessly, transparently and dynamically when needed, such as the power Grid supplies electricity to end users. The Grid originated from trying to solve the information and computational challenges of science [1].

Resource discovery and as a result also service discovery is an important issue for the Grid in answering the questions of how a service requester finds the resources/services needed to solve its particular problem and how a service provider makes potential service requesters aware of the computing resources it can offer. Service discovery is a key concept in a distributed Grid environment. It defines a process for locating service providers and retrieving service descriptions. The problem of service discovery in a Grid environment arises through the heterogeneity, distribution and sharing of the resources in different Virtual Organisations (VOs). The two different approaches implemented in the early stages of the Grid software (GLOBUS toolkit, GT [2]) were:

- Monitoring and Discovery Service (MDS),
- Grid Information Service (GIS).

Although these approaches deal only with resource discovery, service discovery can be seen as an extension of resource discovery.

The MDS [3] was initially designed as a centralised way to obtain Grid service information via an LDAP (Lightweight Directory Access Protocol) server. Later designs in MDS-2 have moved to a decentralised approach where Grid information is stored and indexed by index servers that communicate via a registration protocol [4]. Users can then query directory servers. The assignment of content to servers and the overlay topology of those servers is done in an ad hoc fashion.

GIS is a service that allows storing information about the state of the Grid infrastructure [5]. One approach for describing the data is to use a hierarchical model. This is the approach which is currently in place as GISs have been built on top of directory services. The question arises whether these systems

* Corresponding author. Tel.: +44 29 208 79184; fax: +44 29 2087 4598.
E-mail address: simone.ludwig@cs.cardiff.ac.uk (S.A. Ludwig).

1570-8268/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2005.04.001

and the hierarchical model will provide sufficient performance and expressiveness. An alternative solution is to use a relational data model, which arguably is more difficult to implement and scale, but allows for more expressiveness with a relational query language.

Due to the lack of expressive and efficient matchmaking in Grid environments Condor [6] was used. Condor which is used for high-throughput computing is a matchmaking framework which was developed with classified advertisement (ClassAd) for solving resource allocation problems in a distributed environment with decentralised ownership of resources [7]. This framework provides a bi-lateral match where both resource providers and consumers specify their matching constraints, e.g. policy and requirements. A symmetric requirement is then evaluated for each request–resource pair to determine whether there is a match or not.

The Open Grid Services Infrastructure (OGSI) [8] defines a set of conventions and extensions on the use of Web Service Definition Language and XML Schema to enable stateful Web services. It introduces the idea of stateful Web services and defines approaches for creating, naming, and managing the lifetime of instances of services; for declaring and inspecting service state data; for asynchronous notification of service state change; for representing and managing collections of service instances; and for common handling of service invocation faults. Recently, the WS-Resource Framework (WSRF) [9] was proposed as a refactoring and evolution of OGSI aimed at exploiting new Web services standards, specifically WS-Addressing, and at evolving OGSI based on early implementation and application experiences. WSRF retains essentially all of the functional capabilities present in OGSI, while changing some of the syntax (for example, to exploit WS-Addressing) and also adopting a different terminology in its presentation.

Until recently, research on Grids has focused on designing and building Grid middleware that addresses the core problem of Grids which are resource management and services in a distributed environment. Such services include security and data management. Argonne National Laboratory (ANL) has developed an open-source Grid middleware called GLOBUS [2] which has become the de facto Grid middleware for research and possibly production purposes. From the evolution of the Grid software it can be seen that it went from a middleware approach, where many different tools were combined in a toolbox, to a service-based approach which focuses on application-level issues. The approach proposed in this paper follows this direction by taking this service-based view and presents a framework which is developed on the application level. The approach applies semantics to Grid services and to the applications in order to achieve interoperability within Grid environments. The interactions such as service requests with services from the applications and the Grid are matched semantically. As there are many different Grid implementations and applications, which want to make use of the Grid, available, therefore there is a need for semantics to make them interoperable with each other. In order to connect applications such as the High Energy Physics (HEP) experiments to the Grid two interoperability layers are necessary. One interoperability layer is attached to the application layer and the other to the collective layer. The first interoperability layer serves as a dictionary, allowing the different HEP applications to specify their service needs in their "own" application context. The second interoperability layer allows the definition of semantic service description in order to allow a more flexible and dynamic service discovery process [10].

This paper is organised as follows. In Section 2 related efforts are summarised and the differences to the proposed approach are discussed. Section 3 gives an introduction to the background of semantics and ontologies. The framework of the semantic service discovery approach for Grid environments with a detailed description of the components is shown in Section 4. Section 5 presents a portal prototype implementation and explains the tools used. In Section 6 an enhancement of the matchmaking process by means of a similarity metric is done. Section 7 presents an evaluation of the system by an introduction of a similiarity metric and finally, Section 8 concludes this paper.

2. Related efforts

During the past few years lots of effort and research have been placed in the field of resource matching which are described in the following paragraphs. The different approaches are based on resource matching, resource mapping and selection, and developing infrastructural middleware.

myGrid [11] is a multi-organisational project aiming to develop the necessary infrastructural middleware (e.g. provenance, service discovery, workflow enactment, change notification and personalisation) that operates over an existing Web services & Grid infrastructure to support scientists in making use of complex distributed resources. The myGrid project is to provide access to its bioinformatics archives and analysis tools through Web service technologies using open specifications.

Deelman et al. [12] address the problem of automatically generating job workflows for the Grid. They have developed two workflow generators. The first one maps an abstract workflow defined in terms of application-level components to the set of available Grid resources. The second generator takes a wider perspective and not only performs the abstract to concrete mapping but also enables the constriction of the abstract workflow based on the available components. The system operates in the application domain and chooses application components based on the application metadata attributes.

The GRIP (Grid Interoperability Project) [13] addresses the problem of resource description in the context of a resource broker being developed, which is able to broker for resources described by several Grid middleware systems, GT2, GT3 and Unicore. The approach is based on a semantic solution to resource description. The semantics of the request for resources at an application level needs to be preserved in order to allow appropriate resources to be selected by intermediate agents such as brokers and schedulers. The matchmaking is based on a semantic translation of the different resource description schemas.

Tangmunarunkit et al. [14] have designed and prototyped an ontology-based resource selector that exploits ontologies, background knowledge, and rules for solving resource match-

ing in the Grid to overcome the restrictions and constraints of resource descriptions in the Grid. Traditional resource matching, as done by the Condor Matchmaker [6] or Portable Batch System [15], matchmaking is based on symmetric, attribute-based matching. In order to make the matchmaking more flexible and also to consider the structure of VOs the framework consists of ontology-based matchmakers, resource providers and resource consumers or requesters. Resource providers periodically advertise their resources and capabilities to one or more matchmakers using advertisement messages. The user can then activate the matchmaker by submitting a query asking for resources that satisfy the request specification. The query is then processed by the TRIPLE/XSB deductive database system [16] using matchmaking rules, in combination with background knowledge and ontologies to find the best match for the request.

All these related projects are trying to overcome the interoperability problem which Grid systems face. However, all of them, except of the myGrid project and the abstract workflow mapping project, are concerned with applying semantics to resources in order to have a more powerful matchmaking technique. The myGrid project focuses on the application-level by providing a platform with existing Web services and Grid infrastructure to support scientists in making use of complex distributed resources, whereas the project of Deelman et al. is concerned of mapping complex workflows onto Grid environments. Although the Grid community has produced a number of middleware systems – Globus, Legion [17] and NetSolve [18], to name a few – many areas of the Grid concept remain to be investigated.

The approach proposed in this paper is also concerned with application-level issues and requirements. The main requirements which have driven the development were high degree of flexibility and expressiveness, support for subsumption and datatypes and a flexible and modular structure implemented with latest Web technologies. The main difference to the approaches proposed by others is the concept of a three-step discovery process consisting of application context selection, services selection and registry selection. It allows to capture the application and Grid services semantics separately and it supports application developers and Grid services developers to register application and services semantics separately. For the discovery process, this separation allows a classification of the application semantics in order to find service descriptions in the Grid services ontology.

3. Background to semantics and ontologies

Ontologies contain categories, lexicons contain word senses, terminologies contain terms, directories contain addresses, catalogs contain part numbers, and databases contain numbers, character strings and BLOBs (BinaryLarge OBjects). All these lists, hierarchies and networks are tightly interconnected collections of signs. But the primary connections are not in the bits and bytes that encode the signs, but in the minds of the people who interpret them. The goal of various metadata proposals is to make those mental connections explicit by tagging the data with more signs. Those metalevel signs themselves have further interconnections, which can be tagged with metametalevel signs. But meaningless data cannot acquire meaning by being tagged with meaningless metadata. The ultimate source of meaning is the physical world that uses signs to represent entities in the world and their intentions concerning them [19].

The so-called Rich Text Format (RTF) is semantically the most impoverished representation for text ever devised. Formatting is an aspect of signs that makes them look pretty, but it fails to address the more fundamental question of what they mean. To address meaning, the markup languages in the SGML (Standard Generalized Markup Language) [20] family were designed with a clean separation between formatting and meaning. When properly used, SGML and its successor XML (Extensible Markup Language) [21] use tags in the text to represent semantics and put the formatting in more easily manageable style sheets. That separation is important, but the semantic tags themselves must have clearly defined semantics. However, most XML manuals do not provide guidelines for representing semantics.

Ontologies are increasingly seen as a key technology for enabling semantics-driven knowledge processing. Communities establish ontologies, or shared conceptual models, to provide a framework for sharing a precise meaning of symbols exchanged during communication. A prerequisite for widespread use of ontologies is a joint standard for their description and exchange.

RDF(S) (Resource Description Framework Schema) [22] is an ontology/knowledge representation language which contains classes and properties (binary relations), range and domain constraints (on properties) and subclass and subproperty (subsumption) relations. RDF(S) is a relatively primitive language, however, more expressive power would clearly be necessary and desirable to describe resources in sufficient detail. Moreover, such descriptions should be amenable to automated reasoning if they are to be used effectively by automated processes [23].

These considerations led to the development of the Ontology Inference Layer (OIL) [24] and later to the design of DAML + OIL [25]. DAML + OIL is a more recent proposal for an ontology representation language that has emerged from work under DARPA's Agent Markup Language (DAML) initiative along with input from leading members of the OIL consortium. DAML + OIL is based on the original OIL language, but differs in a number of ways. DAML + OIL provide a greater interoperability on the semantic level. In this way, DAML + OIL extends the RDF(S) basic primitives for providing a more expressive ontology modeling language and some simple terms for creating inferences. In particular, DAML + OIL has moved away from the original frame-like ideas of OIL and it is an alternative syntax for a description logic.

The question arises how semantics help the service discovery process. Service discovery in Grid environments to date are only based on particular keyword queries from the user. This, in majority of the cases leads to low recall and low precision of the retrieved services. The reason might be that the query keywords are semantically similar but syntactically different from the terms in service descriptions. Another reason is that the query keywords might be syntactically equivalent but semantically different from the terms in the service description. Another

problem with keyword-based service discovery approaches is that they cannot completely capture the semantics of a user's query because they do not consider the relations between the keywords. One possible solution for this problem is to use retrieval based on semantics.

4. Semantic service discovery framework

This section describes the semantic service discovery framework for a Grid environment. It gives a description of the components of the framework and shows how the matchmaking process is done.

4.1. Framework requirements

The fundamental problem the Grid research and development community is seeking to solve is how to coordinate distributed resources amongst a dynamic set of individuals and organisations in order to solve a common collaborative goal. The degree of distribution of an application that can run within such an organisation can vary on a scale that runs from a centralised application that uses network resources, but where control and data resides at one location to an application made up of a number of autonomous components that collaborate to meet some overall application goal. Due to many different implementations of Grid software distributed all over the world there is a need to make these implementations interoperable. This leads to the following requirements of the matchmaking framework. The first five requirements are derived from the necessity of using semantics for the service discovery process and the last two requirements are derived from the need to implement a service discovery framework for Grid environments.

1. High degree of flexibility and expressiveness
   Different advertisers would want to describe their Grid services with different degrees of complexity and completeness. The description tool or language must be adaptable to these needs. An advertisement may be very descriptive in some points, but leave others less specified. Therefore, the ability to express semi-structured data is required.
2. Support for subsumption
   Matching should not be restricted to simple service name comparison. A type system with subsumption relationships is required, so more complex matches can be provided based on these relationships.
3. Support for data types
   Attributes such as quantities should be part of the service descriptions. The best way to express and compare this information is by means of data types.
4. Matching process should be efficient
   The matching process should be efficient which means that it should not burden the requester with excessive delays that would prevent its effectiveness.
5. Appropriate syntax for the Grid
   The matchmaker must be compatible with Grid/Web technologies and the information must be in a format appropriate for a Grid environment.
6. Flexible and modular structure
   The framework should be flexible enough to allow Grid applications to describe their context semantics and Grid services to describe their service semantics in a modular manner.
7. Lookup of matched services
   The framework should provide a mechanism to allow the lookup and invocation of matched services.

Starting from these requirements a framework has been developed which is based on semantic service descriptions and it fulfils the requirements as follows. An important element of semantic matchmaking is a shared ontology. Shared ontologies are needed to ensure that terms have clear and consistent semantics. Otherwise, a match may be found or missed based on an incorrect interpretation of the request. The framework supports flexible semantic matchmaking between advertisements and requests based on the ontologies defined. Minimising false positives and false negatives is achieved with three selection stages in combination with well-defined ontologies. The selection stages are:

- Context selection, where the request is matched within the appropriate application context.
- Semantic selection, where the request is matched semantically.
- Registry selection, where a lookup is performed.

The design of having application and Grid service ontologies separate allows a modular design. Furthermore, it encapsulates the application knowledge from the Grid service knowledge. This allows other applications to specify their application semantics separate from the Grid service semantics. The Grid service ontology is specified by Grid developers and the application ontology is developed by the application users. The matchmaking engine should encourage providers and requesters to be precise with their descriptions. To achieve this, the service provider follows an XML-based description, which is the ontology language DAML + OIL. To advertise and register its services the service requester generates a description in the specified DAML + OIL format. Defining the ontologies and the selection stages precisely allows the matchmaking process to be efficient. Semantic matchmaking is based on DAML + OIL ontologies. The advertisements and requests refer to DAML + OIL concepts and the associated semantics. By using DAML + OIL, the matchmaking process can perform implications on the subsumption hierarchy leading to the recognition of semantic matches despite their syntactical differences between advertisements and requests. The use of DAML + OIL also supports accuracy, which means that no matching is recognised when the relation between the advertisement and the request does not derive from the DAML + OIL ontologies used by the registry, where the lookup of the service is performed.

4.2. Matchmaker description

The semantic matchmaking framework in Fig. 1 consists of service requesters (Grid applications), service providers (Grid

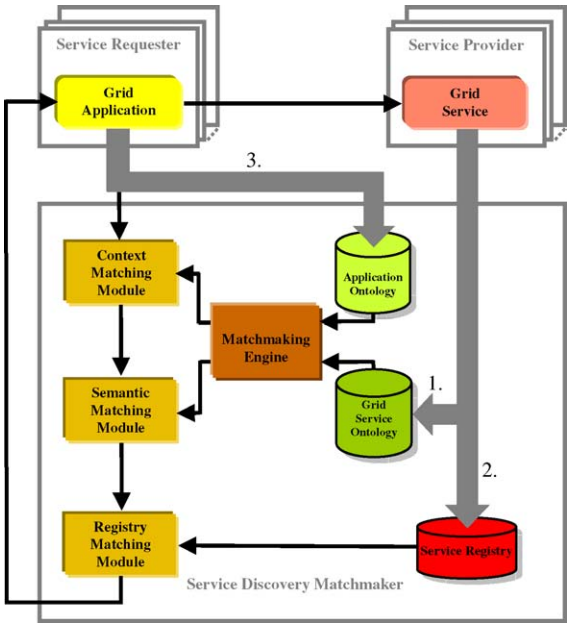

Fig. 1. Semantic service discovery matchmaker – registration.

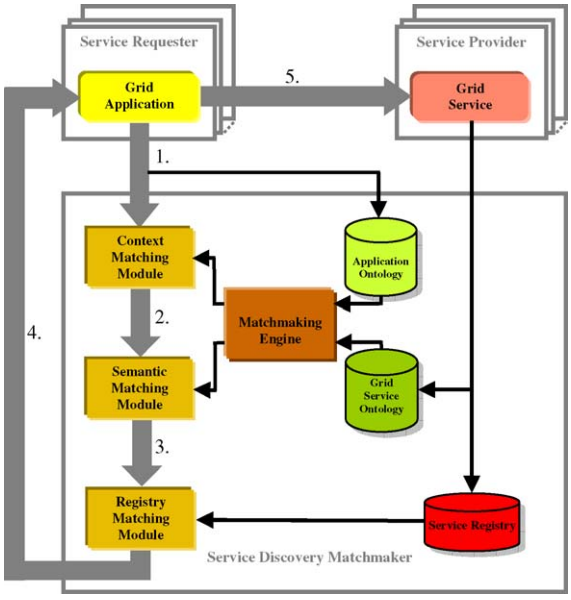

Fig. 2. Semantic service discovery matchmaker – matchmaking.

services) and a service discovery matchmaker. The matchmaking process is designed with respect to the criteria listed in Section 4.1. The processing of a received service request by the matchmaking engine is explained as follows [26]. Depending on the matching modules and the defined application and services ontologies, a semantic match is performed. Every pair of request and advertisement has to go through several different matching modules of the matchmaking process. The final match with the service registry is performed in the registry module. Information is provided to the service requester by sending contact details and related capability descriptions of the relevant service provider.

Fig. 1 shows the interactions of a service registration process. First, the service providers need to register their services for the matchmaking process. The service provider registers its service semantics in the Grid service ontology (1) and the necessary contact details in the service registry (2). Service semantics comprises of a service name, a service description, service attributes (input/output) and metadata information. Furthermore, the service requester specifies the context semantics of the application in the application ontology (3).

The interactions of a service request are shown in Fig. 2. The Grid application sends out a request to the service discovery matchmaker (1). The request has to go through the context matching module first. Here, the request is matched within the appropriate context of the application ontology. This means that depending on the service request, which came from one of the applications, the appropriate context ontology is chosen and the first match is performed. Additional parameters are attached to the request and forwarded to the semantic matching module (2). In this module the semantic match is performed. Semantic matchmaking allows the service request to be matched using the semantics (metadata) of services. Having all necessary semantic data, a service lookup is done using a service registry (3). This lookup information is sent back to the Grid application (4) to be used for the Grid service call (5).

4.3. Matchmaking process

Chosen for the application ontologies of the prototype were the HEP experiments ALICE [27], ATLAS [28], CMS [29] and LHCb [30]. They require huge distributed computational infrastructures to satisfy their data processing and analysis needs and want to access the Grid in order to process their petabytes of data necessary for their experimental evaluations. Interoperability is a main issue for these experiment applications to function with the Grid. The application ontologies were derived from a document about common use cases for the four HEP applications [31]. The services were extracted from the document and structured into four categories which are basic services, data management services, job management services and VO management services. The Grid service ontology was built by defining the related Grid services which are available in the GLOBUS toolkit.

The three matching modules, which are the heart of the matchmaker are described more in detail below. The context matching module allows to match the service request by means of context semantics defined in the application ontologies. The application software of the different HEP applications specifies the service request within their own application context. In this module a mapping from an application service request to a context-based service request is performed. Fig. 3 shows a code fragment of the HEP application ontology. It contains the concept of the application domain specified by classes, datatypes and properties. The matching engine comprises a DAML parser, an inference engine and a defined set of rules in order to reason about the ontologies.

The semantic matching module is responsible for matching the request semantically. This is performed as follows. The Grid services ontology is parsed by a DAML parser. The DAML

```
<daml_oil:Class rdf:ID="CMSJobSubmission">
  <rdfs:comment>Submission of CMS Job.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#CMSJobManagement"/>
</daml_oil:Class>

<daml_oil:DatatypeProperty rdf:ID="CMSEnvironment">
  <daml_oil:domain rdf:resource="#CMSJobSubmission "/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/
>
</daml_oil:DatatypeProperty>

<daml_oil:DatatypeProperty rdf:ID="CMSFileList">
  <daml_oil:domain rdf:resource="#CMSJobSubmission"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/
>
</daml_oil:DatatypeProperty>

<daml_oil:DatatypeProperty rdf:ID="CMSDataSet">
  <daml_oil:domain rdf:resource="#CMSJobSubmission"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/
>
</daml_oil:DatatypeProperty>

rdf:Description rdf:ID="CMSInputFiles">
  <daml_oil:domain rdf:resource="http://
www.cs.cardiff.ac.uk/user/Simone.Ludwig/#CMSFileList"/>
</rdf:Description>

<rdf:Description rdf:ID="CMSConditions">
  <daml_oil:domain rdf:resource="http://
www.cs.cardiff.ac.uk/user/Simone.Ludwig/#CMSFileList"/>
</rdf:Description>

<daml_oil:DatatypeProperty rdf:ID="CMSFileList">
  <daml_oil:domain rdf:resource="#FileList"/>
</daml_oil:DatatypeProperty>

<daml_oil:Class rdf:ID="FileList">
  <rdfs:subClassOf rdf:resource="#CMSJobSubmission"/>
</daml_oil:Class>

...

<daml_oil:Class rdf:ID="ATLASJobSubmission">
  <rdfs:comment>Submission of CMS Job.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#ATLASJobManagement"/>
</daml_oil:Class>
...
<daml_oil:Ontology rdf:ID="">
  <daml_oil:versionInfo></daml_oil:versionInfo>
  <rdfs:comment>This ontology identifies common use cases
for LHC applications to use Grid services.
</rdfs:comment>
</daml_oil:Ontology>
```

Fig. 3. DAML + OIL code fragment of grid application ontology.

```
<daml_oil:Class rdf:ID="JobManagementServices">
  <rdfs:comment>Job submission and
management.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://
www.cs.cardiff.ac.uk/user/Simone.Ludwig/GridServicesOntol
ogy.daml#Ontology"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID="JobSubmit">
  <rdfs:comment>Send job to Grid computing
resources.</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="#JobManagementServices"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID=" JobSubmitLocal">
  <rdfs:comment>Job submission on local
machine.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#JobSubmit"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID=" JobSubmitRemote">
  <rdfs:comment>Job submission on remote
machines.</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#JobSubmit"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID="JobAnalysis">
  <rdfs:comment>Analyse data to produce scientific
results for publication.</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="#JobManagementServices"/>
</daml_oil:Class>

<daml_oil:Class rdf:ID="JobMonitoring">
  <rdfs:comment>Monitor a single job.</rdfs:comment>
  <rdfs:subClassOf
rdf:resource="#JobManagementServices"/>
</daml_oil:Class>

<daml_oil:DatatypeProperty rdf:ID="FileList">
  <daml_oil:domain rdf:resource="#JobSubmit"/>
  <daml_oil:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"
/>
</daml_oil:DatatypeProperty>

...

<daml_oil:Ontology rdf:ID="">
  <daml_oil:versionInfo></daml_oil:versionInfo>
  <rdfs:comment>This ontology identifies common use cases
for LHC applications to use Grid services.
</rdfs:comment>
</daml_oil:Ontology>
```

Fig. 4. DAML + OIL code fragment of Grid services ontology.

parser is capable of parsing DAML + OIL code. The attributes and classes of DAML + OIL describe the concept of the ontology. It characterises the service for advertisement, discovery and matchmaking. The service request is being matched semantically by parsing the ontology. The DAML + OIL code facilitates effective parsing of service capabilities through its use of generic RDF(S) symbols compared to DAML + OIL specific symbols. With a defined set of rules, an inference engine reasons about the value parameters parsed from the ontology. The output parameters of the inference process are forwarded to the registry matching module where the actual match is performed (Fig. 4).

The inference engine is capable of reasoning with DAML + OIL ontologies. By abstracting the behavior it expects from any inference engine, the semantic matching module is able to interact with this engine.

The matching component compares a current rule to given patterns. This set of rules can be divided into two categories. One concerns the reasoning of instances of classes and the other relates to terminological reasoning in order to determine relationships between the classes themselves.

One of the most basic elements of the RDF and DAML languages are the `rdfs:subClassOf` and `daml:subClassOf` statements. These properties are used to specify a subclass relationship between two classes. One of the intuitive notions of this relation is that any instance of a subclass is an instance of the parent class. Utilising the full power of semantic service discovery requires inference on the relationships between classes which is called terminological reasoning. Through DAML's underlying description logic semantics, objects and classes can be automatically compared, contrasted and reasoned based on the input ontologies.

```
(defquery query-for-types
   "Find all types for a given object."
   (declare (variables ?x))
   (PropertyValue
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   ?x
   ?y
)
```

Fig. 5. JESS rule for finding all properties of a defined class.

Java Expert Systems Shell (JESS) was chosen as a rule-based language [32]. If datatypes (in Jess syntax specified as PropertyValue) of a defined class should be found then the `defquery` in Fig. 5 is invoked. The input parameter for the defined class is declared as variable *x* in the query. With queries such as the one shown below, reasoning classes and attributes of the ontology is achieved in order to provide the matching values for the registry selection.

A simple example to show the reasoning behind the applications and Grid services ontologies is given below. The user provides the input parameters such as `CMSInputFiles` and `CMSConditions`. The service request is sent to the context matching module where these parameters are being matched to the class called `CMSFileList`. This is shown in the application ontology in Fig. 3. This class in turn belongs to the resource `FileList`. Having this additional property for the semantic matching module, the request is being matched and results in resource type `JobSubmit` shown in Fig. 4. Subsumption is then used to find the two services which are: `JobSubmit-Local` and `JobSubmitRemote`. These services can then be matched via registry module with the actual appropriate Job submission services for local and remote submission.

5. Implementation of prototype

The Semantic Grid Service Discovery Portal is a portal for service discovery using an ontology-based matchmaking engine. The tool provides six menus which are login, load ontologies, view ontology, search defined service, list all services and logout. The most common steps will be login, loading of ontologies, searching for a defined service and logout. The three matching modules, especially the semantic service discovery lies behind the search for a defined service (Fig. 6). The user is asked to provide up to four search words describing the service s/he wants to search for. The search request goes through the three matching modules. The application specific service request is made first, which is matched with the appropriate context semantics specified in the application ontologies. Then, the semantic matchmaking is performed by parsing and reasoning the Grid services ontology. At last, the match with the provided registry is done and the matched service(s) is/are displayed in a table. The matchmaking engine performs the semantic match of the requested service with the provided services. This allows a powerful and flexible matchmaking process and provides close matches.

As previously shown, for the application and Grid services ontologies DAML + OIL was chosen as it provides a representative notion of semantics for describing services. DAML + OIL

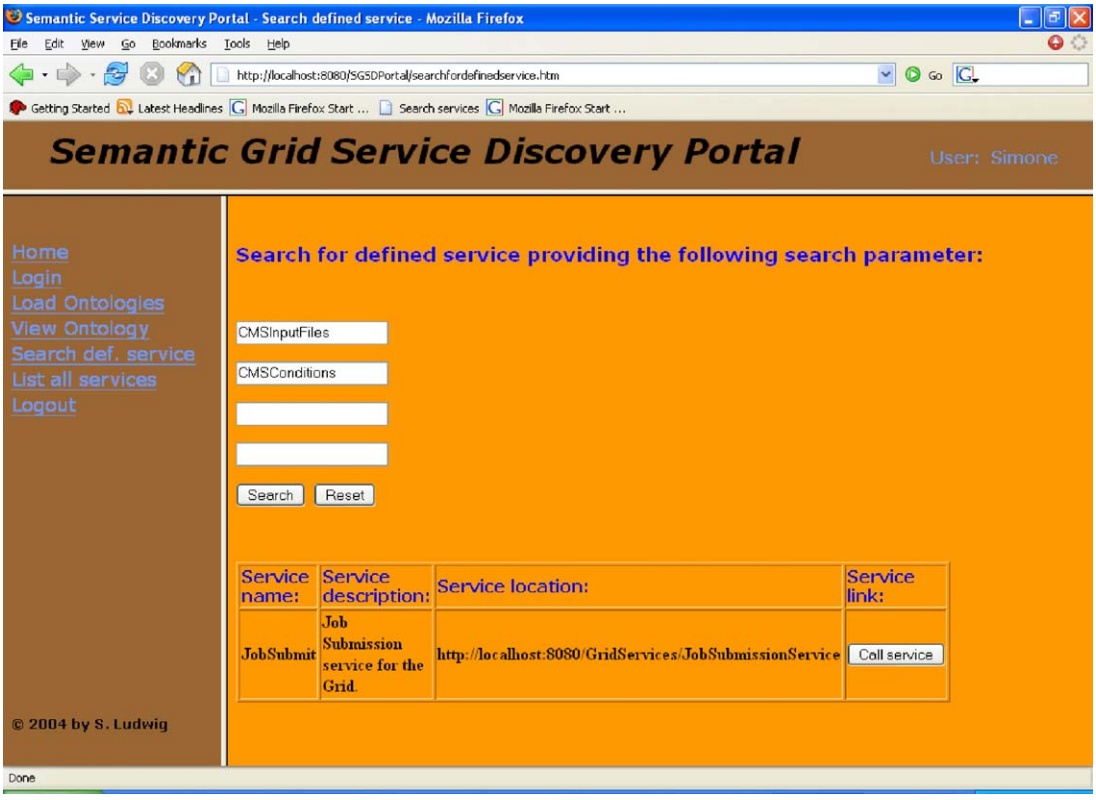

Fig. 6. Semantic Grid service discovery portal.

allows subsumption reasoning on concept taxonomies. Furthermore, DAML + OIL permits the definition of relations between concepts. For the inference engine rules were defined using the JESS language. This API (Application Programming Interface) is intended to facilitate interpretation of information of DAML + OIL files, and allowing users to query on that information. It leverages the existing RDF API to read in the DAML + OIL file as a collection of RDF triples.

This prototype system is based on web services technology standards. The implementation of the web services was done in Java using WSDL (Web Service Description Language), XML (Extensible Markup Language) and SOAP (Simple Object Access Protocol). SOAP and WSDL are designed to provide descriptions of message transport mechanisms in order to describe the interface used by each service.

A service registry, UDDI (Universal Description, Discovery and Integration) [33] was used. UDDI is another emerging XML-based standard to provide a registry of businesses by their physical attributes such as name, address and the services provided. In addition, UDDI descriptions are augmented by a set of attributes that are called TModels. They describe additional features such as the classification of services within taxonomies e.g. NAICS (North American Industry Classification System) or UNSPSC (United Nations Standard Products and Services Code). The UDDI registry is used for the final selection stage which is the registry selection. The actual service is matched with the service request depending on the ontologies loaded.

6. Enhancement of prototype by similarity metric

A drawback related with performing flexible matches is that the matchmaking engine is open to exploitation from advertisements and requests that are too generic in the attempt to maximise the likelihood of matching. For instance, a service may advertise itself as a provider of everything, rather than to be precise with what it does. Similarly, the requester may ask for any service, rather than specifying exactly what it expects. The matchmaking engine can reduce the efficiency of these exploitations by ranking advertisements based on the degree of a match supplied with the request. This is done by an automatic process in order to give an indication of the quality of a match with a similarity metric. This similarity metric allows specifying the degree of flexibility of a match and it also facilitates a ranking of service matches. The similarity algorithm is introduced by implementing a weighting/ranking of service matches, which is an indication for the quality of a match.

An ontology $O_i = \{c_1, \ldots, c_n\}$ contains a set of classes. Each class c_j has an associated set of properties $P_k = \{p_1, \ldots, p_m\}$. Each property has a range indicating a restriction on the values the property can take. An ontology relates more specific concepts to more general ones (from which generic information can be inherited). Such links have been variously named "is a", "subset of", "member of", "subconcept of", "superconcept", etc. Such links are used to organise concepts into a hierarchy or some other partial ordering called "taxonomy". The taxonomy is used for storing information at appropriate levels of generality and automatically making it available to more specific concepts by means of a mechanism of inheritance. The global similarity function $S(c_1, c_2)$ (1) is a weighted sum of the similarity values [34].

$$S(c_1, c_2) = \omega_1 S_1(c_1, c_2) + \omega_2 S_2(c_1, c_2) + \omega_3 S_3(c_1, c_2) \quad (1)$$

The similarity function $S_M(s_1, s_2)$ (2) for the matchmaker is derived from Eq. (1) whereby ω_a, ω_d, and ω_m are weights of the similarity values for attributes, descriptions and metadata descriptions and s_1 and s_2 are the service description and the service request respectively. The final weights ω_a, ω_d and ω_m are functions of the probability of a type of feature with respect to the probability of the other two types of features (3a–c). The similarity of a service for the Semantic Grid Matchmaker contains service attributes, a service description and metadata information. For each keyword if matched with a semantic description of the service (attributes, service description or metadata information), the similarity of a service request with the service provided is calculated by using weights. The weights for the attributes, service description and metadata information can be chosen depending on the quality of expression for each part of the service. This means that e.g. service attributes are given a higher weight value than the service description as attributes are more likely to express the nature of a service to be matched with than the service description. The similarity values of all matched service descriptions are then aggregated to a similarity value which represents the overall similarity between a service request and a service.

$$S_M(s_1, s_2) = \omega_a S_a(s_1, s_2) + \omega_d S_d(s_1, s_2) + \omega_m S_m(s_1, s_2) \quad (2)$$

$$\omega_a = \frac{P_a}{P_a + P_d + P_m} \quad (3a)$$

$$\omega_d = \frac{P_d}{P_a + P_d + P_m} \quad (3b)$$

$$\omega_m = \frac{P_m}{P_a + P_d + P_m} \quad (3c)$$

whereby a: attributes, d: description and m: metadata.

An advertisement matches a request, when the advertisement describes a service that is sufficiently similar to the service requested [35]. The problem of this definition is to specify what "sufficiently similar" means. Basically, it means that an advertisement and a request are "sufficiently similar" when they describe exactly the same service. But this definition is too restrictive, because providers and requesters have no prior agreement on how a service is represented and additionally, they have very different objectives. A restrictive criterion on matching is therefore bound to fail to recognise similarities between advertisements and requests. It is necessary to allow the matchmaking engine to perform flexible matches, those that recognise the degree of similarity between advertisements and requests in order to provide a softer definition of "sufficiently similar". Service requesters should be allowed to decide the degree of flexibility that they grant to the system. If they allow little flexibility, they reduce the likelihood of finding services that match their requirements, which means, they minimise the false positives while increasing the false negatives. On the other hand, by

```
// Collection contains a collection of vectors containing
// matched strings
Collection collectionOfMatches;
...
// Set similarity zero
int similarity = 0;
Iterator iterator = collectionOfMatches.iterator();
while (iterator.hasNext()) {
  Vector matchedStrings = iterator.next();
  for (int i=0; i<matchedStrings.size(); i++) {
    if (matchedStrings.elementAt(i) == serviceName) {
      // Exact match
      similarity = 1;
      break; //Exit for
    }
    else {
      if (matchedStrings.elementAt(i) ==
      serviceAttribute) {
        similarity = similarity +
        weightAttribute*(1/numberOfAttributes);
      }
      if (matchedStrings.elementAt(i) ==
      serviceDescription) {
        similarity = similarity +
        weightDescription*(1/numberOfDescriptions);
      }
      if (matchedStrings.elementAt(i) ==
      serviceMetaDataDescription) {
        similarity = similarity +
        weightMetaData*(1/numberOfMetaDataDescriptions);
      }
    }
  }
}
return similarity;
```

Fig. 7. Implemented similarity algorithm.

increasing the flexibility of a match, they achieve the opposite effect, that is, they reduce the false negatives at the expense of an increase of false positives. This needs to be carefully considered and balanced with the algorithm proposed.

In general, the algorithm has to evaluate the similarity of its arguments based on their degree of integration. The algorithm needs to be implemented according the derived equations (2) and (3). All search parameters provided by the user can be a service name, a service attribute, a service description or a metadata description. If one parameter is matched e.g. with a service attribute it still will be used for the search of other services. Furthermore, the algorithm needs to consider how many service attributes a service provides in total.

Fig. 7 shows a fragment of the similarity algorithm implementation. First a collection of matched values according to a service needs to be created. Each matching attribute belonging to one service is put in a vector and all vectors containing matched values are put into a collection. Then the calculation of the similarity value begins with the while loop, where each vector containing matched values is being calculated. For each vector there is one similarity value calculated at the end, so that for each matched service this value can be displayed. The matched services get then listed in the portal (Fig. 8) according to the similarity value which shows the ranking of the services.

In order to demonstrate how the similarity algorithm works seven services were chosen to query from the Grid service ontology. The search query consists of the search words `DS`, `FileList`, `job`, `submission` and `jobsubmission`.

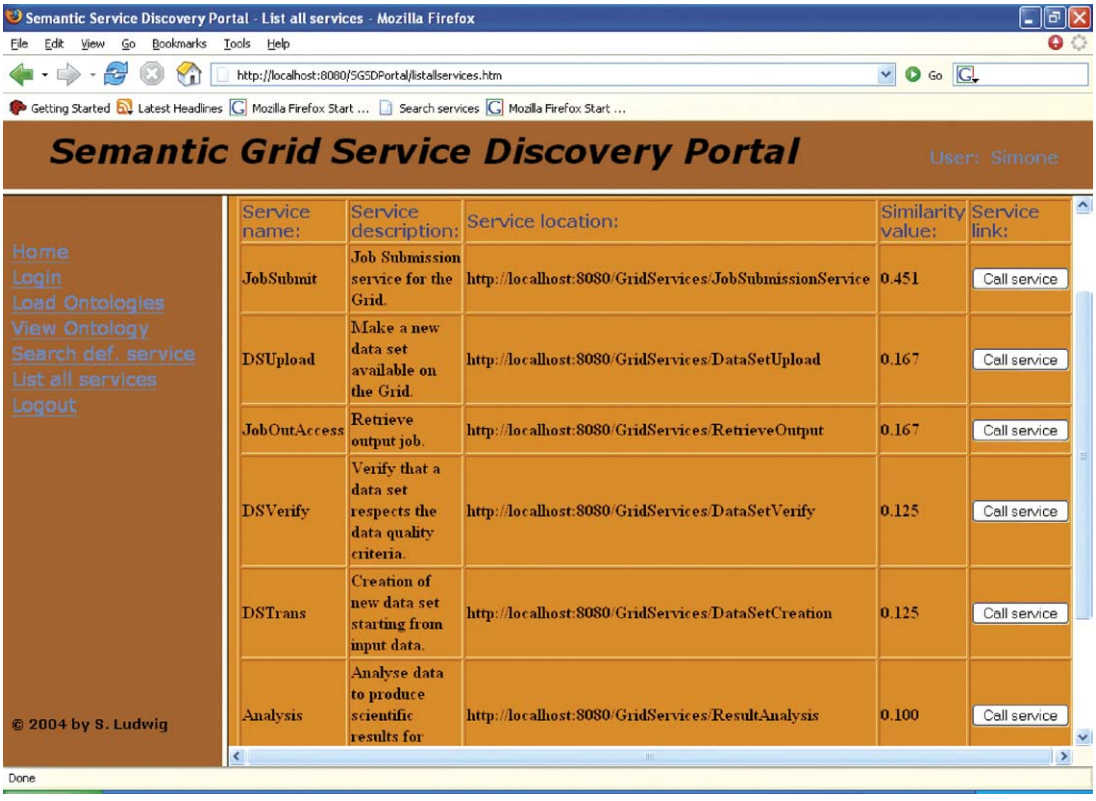

Fig. 8. Similarity values for matched services.

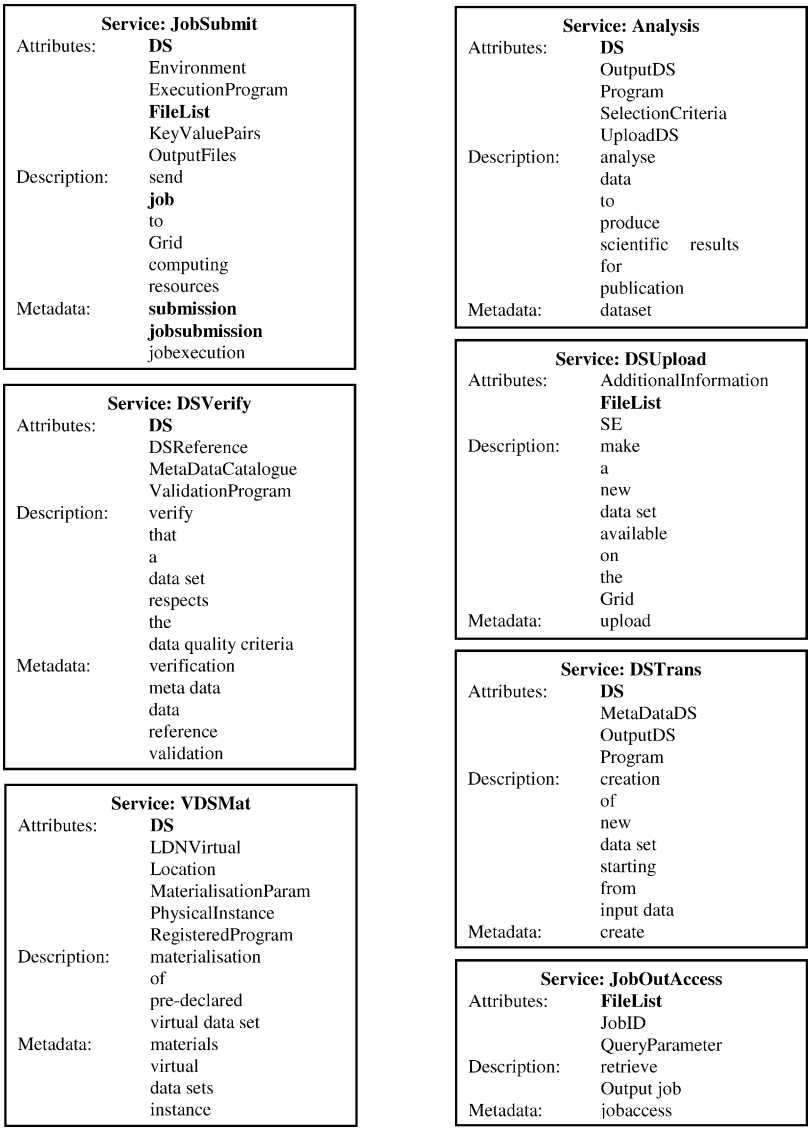

Fig. 9. Semantic search example – similarity metric.

In Fig. 9 these search words are written in bold to highlight the matched parameters of each service. The figure shows the services which are part of the semantic match and get matched by the matchmaker. The attributes and descriptions of the service are given in the boxes as defined in the Grid service ontology. The portal in Fig. 8 shows the calculated similarity values and lists the matched services ranking from the best to the worst match.

As expected, `JobSubmit` was ranked with the highest similarity value. The chosen weights are $\omega_a = 0.5$, $\omega_d = 0.1$ and $\omega_m = 0.4$. These values can be defined depending on the user's preferences or they can be hardcoded within the program. In addition, a threshold value needs to be defined in order to remove false matches from the list of matched services.

7. Evaluation of prototype

The evaluation of the semantic matchmaking modules is done using a qualitative and a quantitative analysis. The qualitative analysis discusses the advantages and disadvantages and suggests the potential for further improvements. The quantitative analysis is to show that the prototype implementation satisfies the performance requirements as applied in real-world applications and most importantly to show the quality improvement of the matchmaking. Performance measurements were conducted to investigate the overall performance of the prototype and in particular the performance scalability of the semantic matchmaking module regarding an increase of the complexity of the ontology and an increase of the complexity of the rules implemented [36].

7.1. Qualitative analysis

In order to analyse the system, the advantages and disadvantages of the matchmaker system are discussed.

Beginning with the advantages, the semantic matchmaking approach allows a powerful and flexible service discovery process as it uses semantic service descriptions. Using semantics

allows to reason on values which is not only based on type reasoning, it furthermore allows subsumption reasoning. This means that the service discovery is very powerful as not only a service name match is performed. Services which would have never been found with the "syntactic" service discovery method can get discovered. Furthermore, the prototype allows customisation of the service discovery process as it provides a selection of service matches to the user. The user can select which service seems to be the appropriate one or if the expected "right" match is not returned, the user can specify another service request with different search parameters. The semantic approach facilitates interoperability as the service properties are defined and specified in associated ontologies. The specification of services and their relations are stored in an ontology which in turn represents the domain knowledge. Unnecessary re-writing of code or interface wrapping does not need to be done in order to make systems interoperable. The development and maintenance is much easier due to the modular structure. Whenever a service needs to be added only an entry in the ontology needs to be added and nothing else. The rules defined in the reasoning engine do not need to be modified and the service discovery process is not affected at all when adding services.

The disadvantage of this semantic approach is that the semantic service discovery is more time consuming due to the additional context and semantic matching modules. This is investigated and evaluated with performance measurements in the following section.

7.2. Quantitative analysis

The aim of the quantitative analysis is to investigate how the prototype system scales by means of performance measurements. Measured is also the performance of an ontology increase and a rule increase.

The semantic service discovery prototype and the additional performance measurement code were stored on a laptop. For the scaling performance analysis only the semantic matchmaking module was considered as the context matchmaking module shows the similar performance reduction. The measurements were done querying all properties of the ontology used. The search request was specified to find all objects in the ontology. Table 1 presents the average measured time results for the three matching modules. The context matching module is roughly 10 times slower than the registry selection but 1.6 times faster than the semantic matching module. The semantic matching module is 17 times slower than the registry matching module. From this table it is revealed that the time consuming part is the semantic and the context matching modules due to the parsing of their ontologies and the rules applied. The total time result of the matchmaking process of the prototype is an average of 3953 ms. This is the time the user has to wait until a service request is performed and the matched services list is returned.

Table 1
Comparison of matching modules

Matching module	Average time (ms)
Context	1475
Semantic	2338
Registry	140

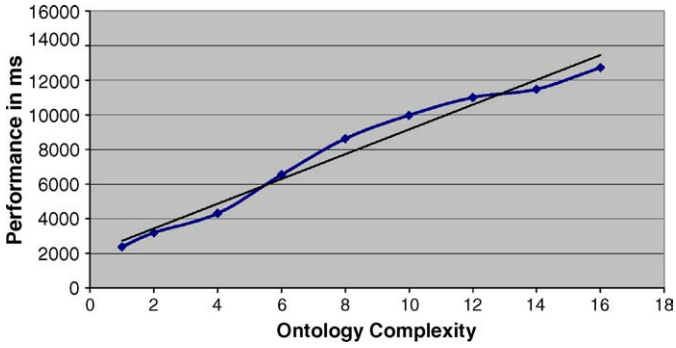

Fig. 10. Performance vs. ontology complexity.

As expected, matching service requests semantically leads to a decrease in performance. If only a service name match was desired, only the registry matching module would be necessary. This would result in a much faster match of around 27 times, but in turn it would not perform a semantic match.

7.2.1. Ontology complexity

Performance measurements for the semantic matchmaking process were conducted in order to see how the time over the complexity of an ontology increases.

The following conditions were met. All nine ontologies of different complexity levels were placed on the Internet, so that real world measurements could be conducted. The complexity of 1 of the ontology is defined as having 112 elements, thereof 47 classes and 65 data type properties. Complexity 2 is the double amount of elements. Having complexity 16 results in 1792 elements, where 752 are classes and 1040 are data type properties.

Fig. 10 shows the performance measurements of the semantic matchmaking module. The graph shows a linear distribution. The regression line shows the average increase of about 700 ms per increase of complexity of the ontology. There seems to be an offset of about 2000 ms. This is due to the instantiation and resetting of the reasoning engine and the rules and queries applied.

7.2.2. Rule complexity

The rule complexity measurements are conducted by increasing the complexity of the rules, queries and facts. Table 2 shows a summary of the `defqueries`, `defrules` and `deffacts` used in the semantic service discovery prototype.

The standard queries include 17 queries, the standard base rules comprise of 9 rules and 57 facts. The basic constraint and

Table 2
Queries, rules and facts

	Defqueries	Defrules	Deffacts
Standard queries	17		
Standard base rules		9	57
Basic constraints		2	
Standard facts			8

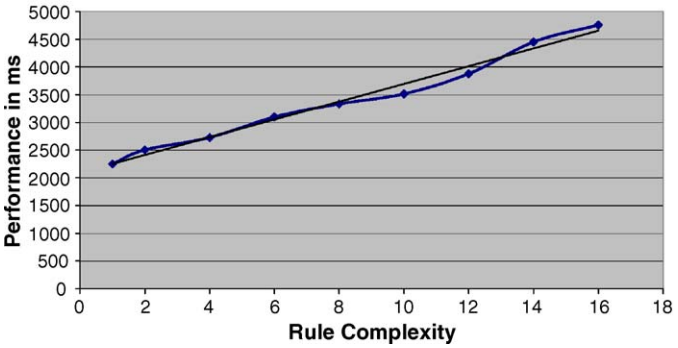

Fig. 11. Performance vs. rule complexity.

standard facts contain of 2 defined rules and 8 facts respectively. Rule complexity 1 is defined having the values as shown in Table 1. Complexity 2 is measured taking the double amount of queries, rules and facts specified. For rule complexity 16 there are 272 queries, 176 rules and 1040 facts which were applied.

Fig. 11 shows a linear distribution. The regression line reveals an average 160 ms performance loss per increase of rule complexity. The offset of the regression line is around 2080 ms, which is roughly the same as shown in Fig. 10 where 2000 ms were measured.

7.2.3. *Precision and recall*

Precision is the fraction of advertised services which is relevant. The highest number is returned when only relevant services are retrieved. Recall is the fraction of relevant services which has been retrieved. The highest number is returned when all relevant services are retrieved.

Three queries were chosen, the first retrieving 8 services whereby 5 services were relevant and 3 were not retrieved. The second query retrieved 5 services whereby 4 were relevant and 1 service was not retrieved. The third query retrieved 4 services whereby 3 were relevant. Fig. 12 shows the precision and recall rates. The precision values apart from one exception are in the range of 0.67–1. This indicates relatively high precision rates of the prototype.

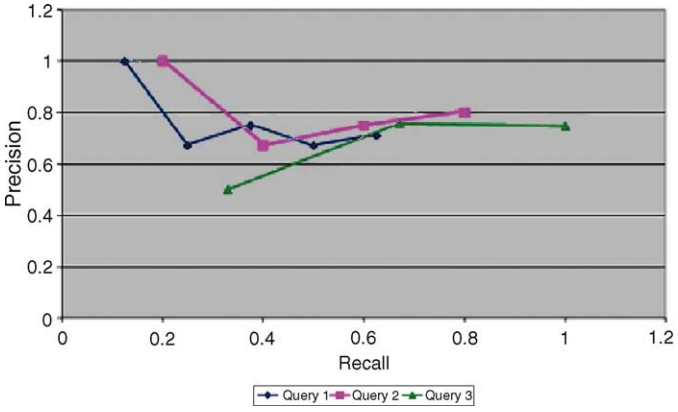

Fig. 12. Recall and precision.

8. Conclusion

The Semantic Grid Matchmaker achieves interoperability for service discovery by using a semantic matchmaking approach. The requirements which have driven the development were high degree of flexibility and expressiveness, support for subsumption and datatypes and a flexible and modular structure. This approach enables a more flexible and dynamic matchmaking mechanism based on semantic descriptions stored in ontologies. The separation of application and Grid service knowledge provides a modular, flexible and extensible structure. It allows the Grid service developer and the application developers to specify their domain knowledge separately.

The prototype was built as a proof of concept of the matchmaking framework proposed for Grid environments. It was found that the problem of performing flexible matches is that the matchmaking engine is open to exploitation from advertisements and requests that are too generic. This means that the matchmaking process needs to restrict the return of matches. Only matches that are sufficiently similar to the service request can be accepted. This was achieved with the similarity algorithm implemented. It allows a ranking of service matches and allows restricting matches which are below a certain similarity value. However, there are still a few problems with the algorithm proposed. The service description also contains prepositions or articles such as to, in, of, a etc., which need to be removed as they do not express the functionality of the service and only distort the similarity value. Furthermore, not every attribute or description has the same expressiveness and should be ranked with a different weight value accordingly. This implies that human intervention for the ranking process becomes necessary which is a drawback for the automation of the matchmaking process.

Looking at the performance for the scalability of the prototype, the performance decreases when the complexity of the ontology rises and also when the complexity of the rules rises. The linear increase of rules has a smaller impact on the performance than the linear increase of complexity of the ontology. The reason for that is that the parsing of the ontology, the greater the complexity becomes, takes more time than only increasing the number of rules applied. From the set of measurements taken it can be seen that this semantic matchmaking module does not scale very well. However, semantic matchmaking is performed which allows an increase of finding the appropriate service. How large the performance loss is depends on the complexity of the ontology and the rules defined. A faster reasoning process is desirable and needs to be investigated.

References

[1] C. Goble, The Grid – from concept to reality in distributed computing, Bioinformatics World Article, 2003. http://www.bioinformaticsworld.info/feature3b.html.

[2] The GLOBUS Project. http://www.globus.org/.

[3] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high-performance distributed computations, in: Proceedings of the Sixth IEEE International

Symposium on High-Performance Distributed Computing (HPDC-6), 1997.
[4] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), 2001.
[5] G. von Laszewski, "Quickstart Guide: GIS", 1999. http://www-unix.mcs.anl.gov/~laszewsk/papers/ldap_in_globus/mdsQuickStartGuide pdf.
[6] The Condor Project. http://www.cs.wisc.edu/condor.
[7] M. Solomon, R. Raman, M. Livny, Matchmaking distributed resource management for high throughput computing, in: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, IL, July, 1998.
[8] I. Foster, et al., The physiology of the Grid: an open Grid services architecture for distributed systems integration, Global Grid Forum, June 2002. http://www.globus.org/research/papers/ogsa.pdf.
[9] The WS-Resource Framework. http://www.globus.org/wsrf/.
[10] S.A. Ludwig, et al., A Grid service discovery matchmaker based on ontology description, in: Proceedings of the Second International EuroWeb2002 Conference, Oxford, UK, December, 2002.
[11] The myGrid Project. http://www.mygrid.org.uk.
[12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, A. Lazzarini, A. Arbree, R. Cavanaugh, S. Koranda, Mapping abstract complex workflows onto Grid environments, J. Grid Comput. 1 (1) (2003) 9–23.
[13] Grid Interoperability Project. http://www.grid-interoperability.org/.
[14] H. Tangmunarunkit, S. Decker, C. Kesselman, Ontology-based resource matching in the Grid – the Grid meets the semantic web, in: Proceedings of the First Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPG03). In Conjunction with the Twelfth International World Wide Web Conference 2003, Budapest, Hungary, May, 2003.
[15] The Portable Batch System. http://pbs.mrj.com.
[16] The XSB Research Group. http://xsb.sourceforge.net.
[17] The Legion Project. http://www.cs.virginia.edu/~legion/.
[18] NetSolve/GridSolve Project 2004. http://icl.cs.utk.edu/netsolve/.
[19] J.F. Sowa, Ontology, Metadata, and Semiotics, Conceptual Structures: Logical, Linguistic, and Computational Issues Lecture Notes in AI #1867, Springer-Verlag, Berlin, 2000.
[20] SGML – C.M. Sperberg-McQueen and Lou Burnard, Chicago, Oxford, A Gentle Introduction to SGML http://www-sul.stanford.edu/tools/tutorials/html2.0/gentle.html.
[21] XML – W3C, Extensible Markup Language (XML) http://www.w3.org/XML/.
[22] RDF – W3C Resource Description Framework (RDF). http://www.w3.org/RDF/.
[23] S. Bechhofer, C. Goble, Towards Annotation using DAML+OIL K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation, Victoria, BC, 2001.
[24] D. Fensel, I. Horrocks, F. van Harmelen, S. Decker, M. Erdmann, M. Klein, OIL in a nutshell, in: Proceedings of EKAW-2000, 2000.
[25] DAML, Darpa Agent Markup Language Program. http://www.daml.org.
[26] S.A. Ludwig, Flexible semantic matchmaking engine, in: Proceedings of Second IASTED International Conference on Information and Knowledge Sharing (IKS), AZ, USA, 2003.
[27] ALICE – A Large Ion Collider Experiment. http://alice.web.cern.ch/Alice/.
[28] ATLAS – A Toroidal LHC ApparatuS. http://atlasinfo.cern.ch/Atlas/Welcome.html.
[29] CMS – Compact Muon Solenoid. http://cmsdoc.cern.ch/cms/outreach/html/index.shtml.
[30] LHCb – Large Hadron Collider. http://lhcb-public.web.cern.ch/lhcb-public/default.htm.
[31] HEPCAL RTAG Report, Common Use Cases for a Common Application Layer (HEPCAL), LHC Grid Computing Project, LHC-SC2-20-2002.
[32] JESS, Java Expert Systems Shell. http://herzberg.ca.sandia.gov/jess/docs/61/index.html.
[33] UDDI Technical White Paper. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.
[34] M.A. Rodriguez, M.J. Egenhofer, Determining semantic similarity among entity classes from different ontologies, IEEE Trans. Knowledge Data Eng. 5 (2) (2003).
[35] M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara, Semantic matching of web services capabilities, in: Proceedings of the International Semantic Web Conference, 2002.
[36] S.A. Ludwig, Evaluation of a semantic Grid service discovery prototype, in: Proceedings of Eighth World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, USA, July, 2004.