P5.2.2
Service Binding and Access Model Registration

Workpackage:	5	Grid Dynamics
Author(s):	Justin Ferris and Mike Surridge	IT Innovation
Authorized by	Mike Surridge	IT Innovation
Doc Ref:	P5.2.2	
Reviewer	Dora Varvarigou	NTUA
Reviewer		
Dissemination Level	Public: equivalent to PU in DoW	

Date	Author	Comments	Version	Status
2006-03-03	Justin Ferris	Initial version for internal review	0.1	Draft
2006-03-06	Mike Surridge	Reformatted to include copyright statements. Reorganised sections, added architectural questions, etc. Incorporated final input from JF.	0.2	Release candidate
2006-03-16	Justin Ferris	Changes to address reviewers' comments, including further text contrasting Grimoires and WSRF-SG, from MS.	0.3	Release candidate
2006-03-17	Justin Ferris	Typos corrected.	0.4	Release candidate
2006-04-03	Justin Ferris	Updated version number to 1.0	1.0	Released

Executive Summary

This document describes project outputs P5.2.2 from NextGRID WP5, concerning the registration of service dynamic behaviour, to enable dynamic adaptation of service consumers at run time.

Registries are fundamental components of service oriented architectures. Current developments in NextGRID suggest a requirement for semantic registry technology in addition to more traditional service registries. The OWL-S ontology language has been extended in NextGRID with the aim of specifying a Workflow and Service Ontology (OWL-WS) that is able to effectively represent dynamic workflows. A registry capable of supporting publication and query of OWL-WS descriptions of workflows and services is required to support the evolving NextGRID architecture and workflow model.

A review of prominent candidates for satisfying semantic registry requirements was undertaken. Based on functionality, security and robustness, it was concluded that a registry called Grimoires from the UK OMII Managed Programme, which supports semantic service descriptions using an extension of the UDDI registry specification, was suitable for further evaluation and development.

Experiments involving publication and query of descriptions of Grid applications have been conducted to evaluate Grimoires query capabilities and other functionality. A client registry interface was designed so that NextGRID components can abstract semantic registry functionality. Implementations of the interface have been produced for Grimoires, as well as for a prototype semantic registry web service, developed during this work to assist evaluation of semantic query technology. Functional evaluation of Grimoires revealed some areas for improvement, including better query support and support for adding custom reasoning so that inference can be performed in the server and more powerful query achieved.

It is expected that NextGRID environments will be highly dynamic with regard to changing populations of published services. Initial evaluation of Grimoires publication and query performance has been conducted in light of this and to highlight areas in need of improvement. Whilst publication and query times are high, it was found that that they scale well with increasing number or published services. Also, some optimisations have been suggested that should improve query and publication performance.

It is concluded that Grimoires is a good starting point for a semantic registry. It is proposed that the functional and performance improvements identified here are addressed, so that it’s able to fulfil the role of a semantic registry in the architecture and support NextGRID’s dynamic workflow model.

Table of Contents

1 Introduction

This report constitutes Deliverable P5.2.2 “Service Binding and Access Model Registration” under EC IST Project 511563 “The Next Generation GRID” (NextGRID) [1]. It describes the output from WP5.2.2, during the six month period starting at the beginning of September 2005 and ending at the end of February 2006.

NextGRID WP5 is concerned with dynamic aspects of Grid systems: how Grid services and clients interact to enable agile federation of resources, services and users between disparate domains. Its goal is to investigate these issues using experimental software. WP5.2 is concerned with dynamic discovery, and its support using registries and related technologies. Registries are fundamental components of service oriented architectures. Current developments in NextGRID suggest requirements for semantic registry technology in addition to more traditional service registries.

The key objectives for the work presented here were:

- identification or development of a semantic registry technology for use in NextGRID
- provision of a semantic registry implementation and programming API to support validation of the Grid VIM and workflow model for experiments conducted in WP5.3 (see below)
- and initial performance profiling of the semantic registry of choice, to identify performance issues expected with semi-structured semantic service descriptions
- resolving some architectural questions raised by the conceptualisation task in NextGRID WP1.

This report is structured as follows. Aspects of the NextGRID architecture that present requirements for semantic registries are considered in this section, before review of registry technology and candidate semantic registries in particular. Section 2 describes objectives, including the relationship of the work to the NextGRID architecture, including the Grid Virtual Infrastructure Model, and highlights the NextGRID architectural questions addressed by the work. Section 3 covers the relationship to the workflow components of the Grid VIM, and the requirements placed on the semantic registry so it could be used in workflow experiments. Section 4 covers the available technologies and specifications considered, including the chosen candidate Grimoires, which supports semantic service description as an extension of UDDI. Section 5 covers the design of the service dynamics registry and the experiment used to validate it, while Section 6 documents the results of functional and performance testing.

Finally, Section 7 provides a summary of the conclusions, which are that RDF and OWL-S or similar languages can be used to describe service dynamics, that these descriptions can be registered and discovered using the Grimoires extensions on UDDI, but that further work is needed to extend the query languages and performance provided by the Grimoires implementation. This section also summarises the answers to the NextGRID architectural questions addressed.

2 Objectives

2.1 NextGRID architecture

The overall objective of the NextGRID project, as described in its Description of Work [1], is to define the architecture for the next generation of Grid middleware.

The architecture is evolving, but the NextGRID Architecture Board has provided a unifying architectural vision and roadmap for its development, and this has been documented in a previous report [2]. The NextGRID goals include broadening the use of academic and research Grids to include applications from the business world. The overall vision includes a secure Grid that supports dynamic virtual organisations (VOs), is viable for business use, and can adapt to different organisations and changes in business policy.

In such a dynamic environment, service discovery cannot be performed out of band and therefore explicit support must be provided by the infrastructure. It's expected that the set of services and applications provided in a NextGRID environment will be large and ever changing. Any discovery capability will have to meet the challenges inherent in this environment. The query rate is likely to be high, both because it's expected that there will be a large number of clients, and because the ever changing population of services will likely mandate a high rate of query by individual clients.

Another challenge is the way of describing and finding services. Registries usually contain certain details of discoverable services. Service look-up happens by querying the registry according to a set of parameters. With the number of services growing, finding usable service candidates becomes more complicated. There are two solutions to reducing the complexity of discovery. One is the use of semantics, the other the use of standardized descriptions. Here the use of semantic technology is considered.

A primary architectural principal in NextGRID is that relationships between services are governed by bipartite Service Level Agreements (SLAs), and the vision held is that any multilateral collaboration or VO model can be formed using bipartite SLA. This emphasis on SLA implies that rich metadata describing non-functional (and dynamic) service properties are important for service discovery, and powerful query over these properties will be required.

Workflow plays a critical role in NextGRID. Workflow language representations can be used to "soft code" business and application processes; and workflow techniques and their theoretical underpinnings are critical to guide correct composition and orchestration of federations of Grid resources. To support the dynamic Grid environment envisaged, the evolution of adaptive workflow techniques is important. These ideas are being explored and developed in WP5.3 and have produced the concept of a Grid Virtual Infrastructure Model or Grid VIM [3]. The Grid VIM is considered next.

2.2 Grid VIM and workflow model

Service providers may publish the detailed workflows that describe the interactions and preconditions necessary for a client to use their service. However, any two service providers may host different Grid infrastructure and have quite different business policies, and this may be true even when they offer the same service or application functionality to potential clients. This poses a problem for client application and workflow developers when they need their application to achieve a functional goal by binding at runtime to either of two services hosted by different service providers.

The vision of the Grid VIM [3] is that adaptive workflow and dynamic binding to services can facilitate abstraction of both business processes and requisite interactions with Grid infrastructure. It aims to address the problem outlined above using adaptive workflow, semantic discovery and service selection heuristics. Application logic can be captured with abstract "application workflows" that include the functional constraints of the application. During workflow execution, abstract application tasks are resolved to concrete implementations through a process that includes discovery, selection and workflow rewiring, before task execution. The key architectural components of the model are illustrated below.

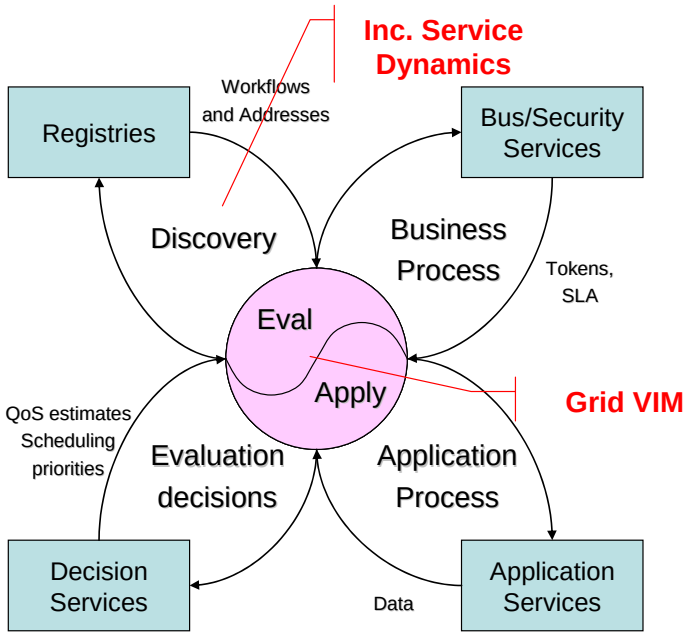

Figure 1. The Grid Virtual Infrastructure Model

The central component is the workflow interpreter, or "Grid VIM for workflow". During workflow execution, an evaluation strategy ("Eval") is used to discover candidate services and workflows that could implement an abstract application step. Evaluation begins by finding candidate replacements using query of one or more registries. Decision services are used to apply candidate selection heuristics and apply local organisational policies that determine which candidate will be selected. After selection of a candidate, replacement is made by rewiring the workflow, and the implementation is executed ("Apply"). This may result in distributed execution of a business or application service.

It's important to note that this model involves more than just late binding of an abstract interface to a concrete implementation at runtime. Abstract tasks may be realised by selection of workflows and data, in addition to services. Furthermore, it's envisaged that decision services will take into account binding decisions made previously, during the course of workflow execution. This may result in a certain amount of "workflow rewiring", such that resources generated during workflow execution can be utilised by late-bound tasks.

In addition to abstracting business and infrastructure workflows, the adaptive workflow and dynamic binding approach of the Grid VIM aims to enable local organisational policy specification and enforcement. This may be achieved largely by organisational control of internal registry contents and imposing candidate selection policies.

2.3 NextGRID architectural questions addressed

The work on semantic registries for service dynamics discovery sought to answer the following questions from the NextGRID Straw Man architecture [4], as updated and extended by the work of the Conceptualisation task from WP1:

Semantic service description

- 1) What semantic information is required to describe NextGRID services?
- 2) What standards for semantic service description exist and are most appropriate for NextGRID?
- 3) What query functionality is required?

Service description

- 1) What further information needs to be in the description (beyond WSDL 1.1)?

Service discovery

- 56.5) Do we use OWL/RDF to describe the semantics?
- 83) Is there a way to integrate process with dynamic policy description? How do we make this into an architectural component?

3 Relationship to Workflow developments

3.1 Workflow description and OWL-WS

Ongoing work in WP5.3 includes evolution and implementation of these ideas and has resulted in development of a workflow language and enactment model for adaptive workflow [3]. The semantic workflow language OWL-WS has been designed for this purpose, which extends the OWL-S[6] service ontology language. OWL-WS is built on RDF [5] and can be used for description of services and workflows, and for annotation of these and other entities with metadata from rich domain-dependent ontologies. OWL-WS is the current language of choice for describing workflows that are evaluated by the workflow enactment component of the Grid VIM. OWL-WS, and OWL-S, are ontology languages that are built on RDF [5]. OWL-WS descriptions are valid RDF and therefore can be represented as a set of triple assertions. The high-level ontology concepts provided by OWL-WS are illustrated below using a UML class diagram.

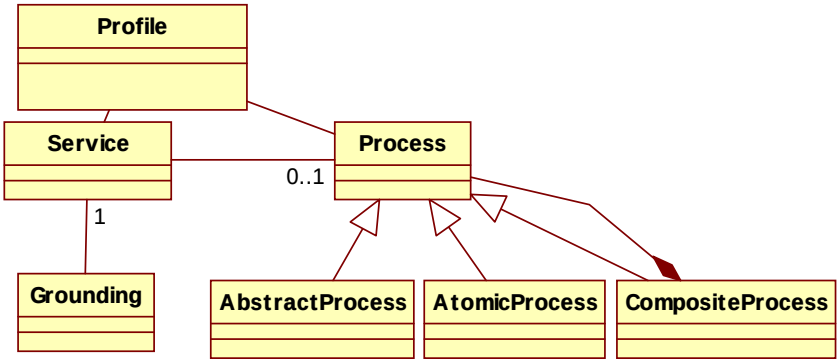

Figure 2. Class structure of the OWL-WS language

A `Service` is used to represent a deployed web service. In both OWL-S and OWL-WS the `Service` entity is largely a placeholder, used to aggregate relevant information relating to a single web service (OWL-S); or service or workflow (OWL-WS).

A `Profile` is used to describe service functionality, including the name of the service, the inputs outputs, preconditions and effects for service execution; and a host of other information, including arbitrary metadata that can be used for any additional service description, including provision of non-functional service properties, for example. Due to the use of RDF-based technologies, metadata can have complex structure and provide rich additional information. One use of Profiles is to publish them in semantic registries to advertise service capabilities.

The original OWL-S definition of a `Process` is that it is the entity that describes the interactions that a client of a web service needs to make. In the simplest case of a single web service operation, the process will be an `AtomicProcess`, whilst stateful services will often be described by a `CompositeProcess` that describes a client workflow. `AbstractProcess` is an OWL-WS extension that is used to represent a process that has functional and non-functional constraints but that isn't realised by a concrete binding to a web service or other executable process. OWL-WS also extends OWL-S by allowing `CompositeProcesses` to represent workflows that orchestrate more than one logical web service.

The `Grounding` describes information for binding services to concrete protocols, endpoints and executable entities. For services described with WSDL, this will typically include reference to service endpoints, and WSDL bindings, port types and messages, in sufficient detail such that an OWL-WS processor can make a service invocation.

3.2 Semantic registry requirements

There are currently two prominent requirements for semantic registry technology in NextGRID, in order that they can be used by a workflow enactor that implements the Grid Virtual Infrastructure Model.

Firstly, publication and query is required for workflows and services described in OWL-WS to support the architecture and functionality of the Grid VIM. This requires rich query over OWL-WS descriptions to identify candidates that satisfy abstract process specifications.

Secondly, a related requirement is that publication and query is needed for semi-structured metadata that may be used to annotate services and workflows with non-functional properties and other concepts of relevance to SLA. These structures are likely to be complex in order to capture concepts from rich ontologies, including those defined in NextGRID and others that model specific application domains. Requirements in other areas, including that of security, are likely to become apparent as work evolves in the project.

The requirements from a semantic registry are thus quite different from other types of service registry, including the WSRF-SG registry developed by NEC [10]. Firstly, semantic registries must store distributed workflows as well as service descriptions, so not everything in a semantic registry will be attached to a service endpoint reference, or have resource properties containing the registered attributes. This makes it difficult to use the WSRF-SG specification in its current form. Secondly, semantic registry entries will be complex, and query processing is likely to be expensive. This means that a semantic registry will need different performance trade-offs between query and update processing, and between scalability and absolute performance. The existing NextGRID WSRF-SG implementation was not specialised in any way for semantic service (let alone workflow) description or query. It was therefore decided to consider it along with other possible starting points against the overall semantic registry requirements, as discussed above.

4 Candidate specifications and technologies

4.1 UDDI

The Universal Description, Discovery, and Integration (UDDI) specification [7] is a well-known registry technology. A UDDI registry allows businesses to advertise themselves and details of their services with a business registration. Such a registration comprises of three kinds of information, sometimes termed “White page”, “Yellow page”, and “Green page” information. “White page” information includes address and contact details. “Yellow page” information includes categorisation information on the business and its services according to standard taxonomies. “Green Page” information includes detailed technical specifications of how to use the services hosted by the business. UDDI specifies a web service API, data model and replication schema and protocol, for sharing data between groups of registries. The UDDI data model is more general than web services, and the key entities in the data model are illustrated in the UML class diagram below.

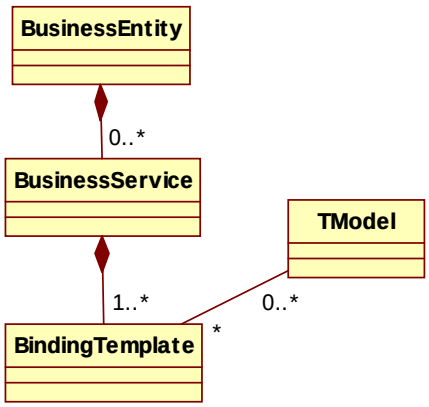

Figure 3. The UDDI data model

The `BusinessEntity` represents all published information about a particular business. The `BusinessService` structure represents a logical service classification, or kind of service supported by a business. Technical descriptions of instances of a particular service, including entry points such as a network endpoint, for example, are accommodated with the `BindingTemplate` entity. The use of `TModel` is quite nebulous. It’s intended to provide technical specifications and metadata about specific service instances. `TModel`’s can be used to represent just about anything but they are often used in UDDI to provide classification taxonomies.

In practice, use of UDDI is quite cumbersome and it is difficult to use for registration and discovery of web services. This is not least because the specification and underlying data model do not mandate a mapping from web service (WSDL) entities to UDDI concepts. However, other fundamental problems include that query is restricted to simple attribute (name/value pair) matching. UDDI provides the notion of the `findQualifer`, to allow, for example, use of wild card expressions and case insensitive matches. These features increase query power, but it’s currently not possible to query a UDDI registry with expressions that contain combinations of comparison and Boolean operators. Support in directory technologies for such queries is essential and semantic query demands more powerful query expression than traditional directory searching. These properties of UDDI make it clear that a vanilla

UDDI registry implementation is inappropriate for the complex data representation and query requirements that have been outlined above.

4.2 WSRF-SG

WS-ServiceGroup (WSRF-SG) provides a way to create collections of service endpoints characterised by their resource properties, and to search these collections using resource property queries [8]. The main use of WSRF-SG is to implement a registry of related web services. WSRF-SG relies on WS-RP [9] for specification of its query mechanism, and this allows for arbitrary query language use and mandates only that XPath query must be supported by compliant implementations. Previous interesting work in NextGRID on WSRF-SG has included development and performance evaluation of an initial implementation of a WSRF-SG service registry [10]. Presently, it seems likely that the limited query support, security issues, and possible scalability problems with the WSRF-SG specification make an implementation unlikely to meet the requirements for a semantic registry.

4.3 Feta

Feta [11] was initially developed in the UK e-science project, myGrid. Feta comprises of two main components: Feta client and Feta engine. The Feta client and associated tools enable users to semantically describe web services and perform domain-specific semantic queries to discover services. Service descriptions are made by annotating an XML representation of the myGrid ontology for services with concepts from domain-dependent ontologies. The Feta engine is more generic than the client and comprises of a web service semantic registry. The engine supports publication of service descriptions in RDF, internal reasoning over domain-dependent ontologies using RDF(S), and semantic query with RDQL [12]. At the present time, the Feta engine lacks database persistence, supporting only in-memory registrations. Service descriptions therefore are lost when the Feta engine process is restarted. Security mechanisms are yet to be addressed also.

4.4 Grimoires

The Grimoires registry [13] is being developed in the UK OMII Managed Programme. It implements UDDI v2 but crucially also extends the UDDI data model and feature set with capabilities for publication and semantic query of metadata annotations of UDDI and web service entities.

Relevant entities in the Grimoires data model are illustrated in the diagram below.

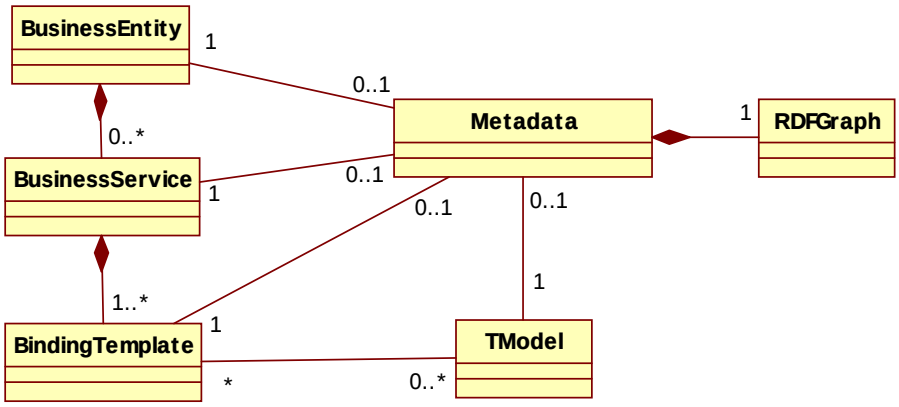

Figure 4. The GRIMOIRES data model

From the diagram, it can be seen that the UDDI data model has been extended such that each entity optionally may be associated with a Metadata entity. This is used to enable arbitrary metadata descriptions of UDDI and WSDL entities (not shown). Grimoires supports different types of metadata descriptions, including use of an RDF graph.

The Grimoires registry supports semantic query using RDQL and has a variety of configuration operations for data persistence, including support for a number of types of relational database or use of in-memory data. Grimoires supports certificate-authentication schemes, SOAP message signing and verification in accordance with WS-Security standards; and fine grained access-control for published entities, including metadata attachments.

Taken together, these features make Grimoires an obvious choice for further evaluation and experiments in service binding and access model registration.

5 Design, development and testing

5.1 Approach

A client registry interface was designed so that NextGRID components can abstract semantic registry functionality and implementations of the interface produced for Grimoires and a custom prototype registry. A straightforward scheme for mapping OWL-WS entities to the Grimoires data model was devised so that OWL-WS descriptions of services, workflows and other entities could be published and queried over. Details of the mapping are provided later in this section.

Simple experiments were conducted to evaluate query functionality and validate semantic registry use by the Grid VIM workflow enactment component. As part of this work, OWL-WS descriptions of GRIA [14] applications were composed and published, in order to test realistic data structures and queries. GRIA is a web service grid middleware created by the University of Southampton and NTUA in the GRIA project, based on components developed by them in GRIA and also in the EC GEMSS and UK e-Science Comb-e-Chem projects.

Finally, initial performance evaluation of Grimoires was conducted to highlight requirements for future work and for comparison with performance results obtained previously for a WSRF-SG registry implementation [10].

5.2 Registry interface design

The interface and implementations produced in this work are illustrated in the UML diagram below.

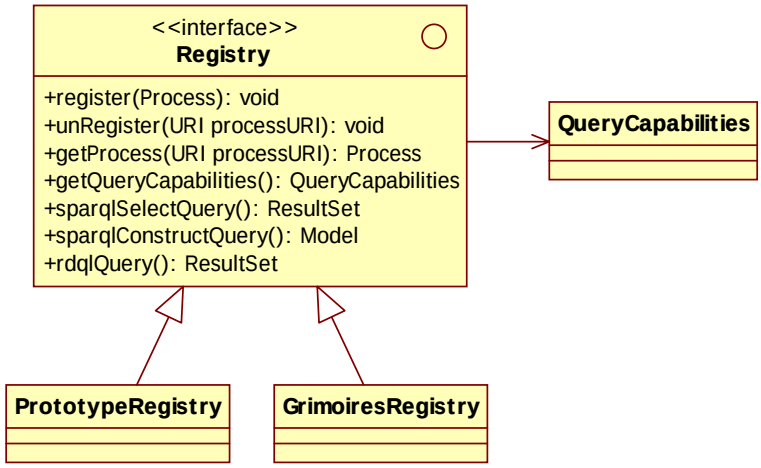

Figure 5. Client-side registry interface design

The `GrimoiresRegistry` class implements the Registry interface and encapsulates the logic required to make web service calls to a Grimoires web service instance, when performing web service operations. As mentioned above, Grimoires supports the now superseded query language, RDQL. SPARQL is a W3C working draft [15] and provides a number of improvements over RDQL for semantic query. To evaluate SPARQL, a custom registry implementation was developed and deployed as a web service. A client stub, `PrototypeRegistry` was also developed to encapsulate client communication with the web service.

The registry interface provides simple methods for registering and de-registering OWL-WS Process entities. The interface presents three different query methods. Two of these are for different forms of SPARQL query and a third is for performing RDQL. It's expected that different registries will provide different degrees of support for these query technologies. The notion of query could be abstracted but for now a `QueryCapabilities` class is used so that clients determine registry compliance with these kinds of query.

5.3 Mapping OWL-WS to Grimoires data model

A mapping between OWL-WS and the Grimoires data model was required so that OWL-WS descriptions could be published and queried over. One such mapping is as follows.

An OWL-WS `Profile` can be mapped to a `TModel`. This fits well because both entities are intended to specify technical specification information. Also, both entities may be associated with zero or more implementations of their technical specification. That is, the `Profile` entity, in OWL-WS, and the `TModel`, in UDDI, maybe associated with zero or more `Process` or `BindingTemplate` entities, respectively. The combination of the OWL-WS concepts `Service`, `Process` and `Grounding` could be mapped to a `BindingTemplate`, which provides the service instance-specific details such as network endpoints.

However, the mapping described above would result in unnecessary publication and query overhead, due to extra client-to-registry communications for both publication and common queries. This is because more than one UDDI entity is being used. Therefore, the simple approach of using a `TModel` for each combination of `Service`, `Profile`, `Process` and `Grounding` was chosen. With this mapping, a single server operation call can be used to retrieve a `Process` with its `Profile(s)`, and publication can be achieved with two calls to server operations: one to publish a `TModel` and another to annotate it with a `Metadata` attachment. The trade-off is that there is potential for duplication of `Profile` data, as each published process will have its own copy of a `Profile`. The simple mapping used can be seen in the class diagram below, which simply illustrates that OWL-WS descriptions are added as metadata attachments to `TModel`s.

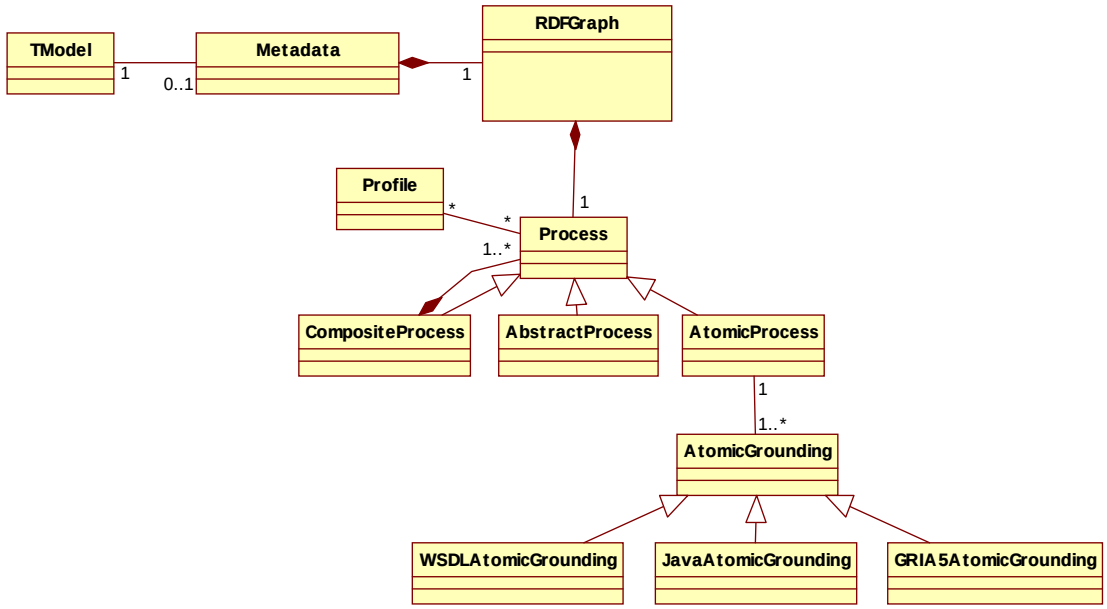

Figure 6. Introduction of dynamic behaviour via the TModel.

5.4 Test system

A simple test system comprising of a client and server on a high-speed local area network was used. A Grimoires registry and prototype semantic registry were deployed as web applications to a Java 2 Enterprise Edition application server running on the server machine. Communication between the test client and the server components was performed using SOAP over HTTP. Grimoires was configured to use for data persistence a MySQL server running on the server machine.

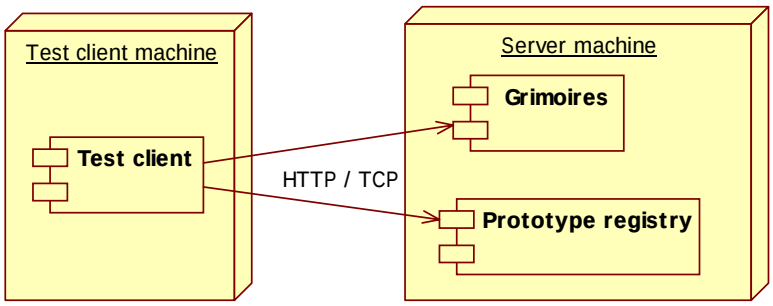

Figure 7. Test deployment

All performance measurements were made in triplicate in similar conditions, having exercised the server components to eliminate bootstrapping delays that would not be applicable in a production environment.

6 Results and analysis

6.1 Functional issues

Experimentation with publication and query of descriptions of GRIA application using Grimoires highlighted a number of functional issues.

6.1.1 Query languages

It was found that queries were limited by the features of RDQL. An example will help clarify the problems encountered. Consider a case in which a client wishes to search a registry for a service that hosts a particular Grid application. The client wishes to use a secure service but has a limited number of options for the security protocols it supports. Furthermore, assume that there is a registry containing descriptions of services and that two of the services match the client's security requirements. The descriptions of the matching services, including concepts from a fictional ontology of security capabilities, are illustrated below.

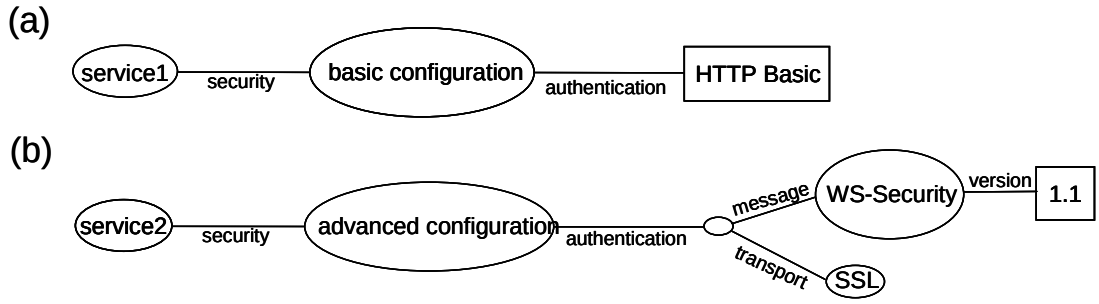

Figure 8. Finding services using RDQL

Here, the description for `service1` states that it uses HTTP basic authentication to provide minimal security by authenticating clients using a username and password. In contrast, the descriptions for `service2` states that authentication is achieved using both WS-Security version 1.1 and SSL. Notice that the differences between the descriptions for the two services are more significant than just having different values for particular properties. The graph structures are quite different.

It isn't possible to express in RDQL a query that would select a service that uses HTTP Basic authentication OR uses WS-Security version 1.1. This is because solution graphs within an RDQL result set have to have the same graph structure.

Query of semi-structured data using RDQL is superior to traditional directory searching, but it was found that the lack of support for disjunction and optional graph patterns quickly causes problems when service descriptions do not conform to rigid schema. RDQL has been superseded by the SPARQL query language for RDF [15]. SPARQL is more expressive and provides, amongst other improvements, direct support for disjunction and optional graph patterns. Using SPARQL, the required query can be expressed easily, and this kind of query was executed successfully using the prototype semantic registry. It's believed that this additional power will be required when ontology and service description work matures in NextGRID. For example, in future we may expect that reference applications developed in WP7 will be described using rich domain ontologies and that these descriptions may be used to annotate services, to advertise that they support a particular application. If this occurs, RDQL will likely prove inadequate for query, unless the annotations are quite constrained.

6.1.2 Data integrity

A second functional issue that arose during testing concerns integrity of data creation and retrieval operations. Grimoires uses a triple store as its underlying persistence model, but it was found that further structure is needed so that following metadata publication, the entire annotation for an entity can be retrieved, updated or deleted.

This is quite a serious observation. The use of RDF allows Grimoires to support very powerful query operations, but it then uses the same mechanism to locate the triples that form part of a discovered service. This makes data retrieval slow, and the presence of duplicated information in multiple profiles can cause the retrieval procedure to fail and return incomplete profiles. Some modifications were made during the experiment to deal with this.

6.1.3 Ontology and reasoning support

A third functional issue is the requirement for direct ontology and reasoning support. There is currently no straightforward way to "plug-in" ontologies and reasoners for OWL-WS so that reasoning can be used when performing semantic queries. This is essential to support queries such as "find all sub types of this service interface", for example.

6.2 Performance results

6.2.1 Registration performance

As described earlier in the report, it is expected that NextGRID environments will be dynamic with regard to changing populations of published services. One useful performance measure, therefore, is assessment of registry publication and update performance with varying numbers of services. Publication performance under different concurrent publication and query loads are also important of course, and these may be considered in future.

The total time taken to publish to the Grimoires registry an OWL-WS description of a typical GRIA application was measured, with varying number of published services. The execution time for the `Registry.register()` method (see above) was measured on the test client by wrapping the method call with efficient performance monitoring code. The results are shown below.

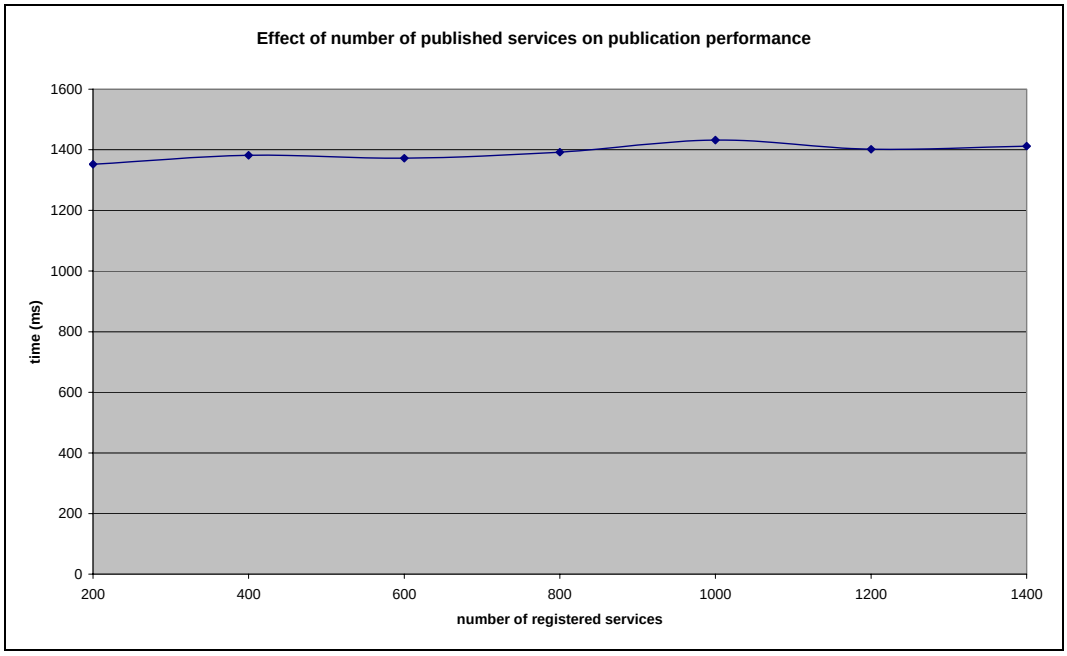

Figure 9. Registration performance

From these results, it's clear that publication times are very high and furthermore there is no clear relationship to the number of published services over the range used (0 - 1400 registrations). This could indicate that the registry is optimised for query or just that there are inefficiencies in the implementation. However, the lack of degradation in performance over this range of published services suggests that overhead of database table indexing on insertion of new data isn't an issue in this range. The initial high publication time is somewhat surprising and warrants future detailed profiling of Grimoires code to identify fine grained areas for optimisation.

6.2.2 Query performance

To assess query times, the client-side `Registry.getProcess()` method was wrapped with performance measurement code and queries over the Grimoires registry were made with varying numbers of published services. Due to the mapping of OWL-WS description to `TModel` metadata annotations, execution of this query by the client requires two network calls to the Grimoires server. The first call causes execution of an RDQL query to find a `Process`, and the second retrieves the metadata attachment that contains the OWL-WS description. These were measured independently, along with the total time. The results are shown and analysed below.

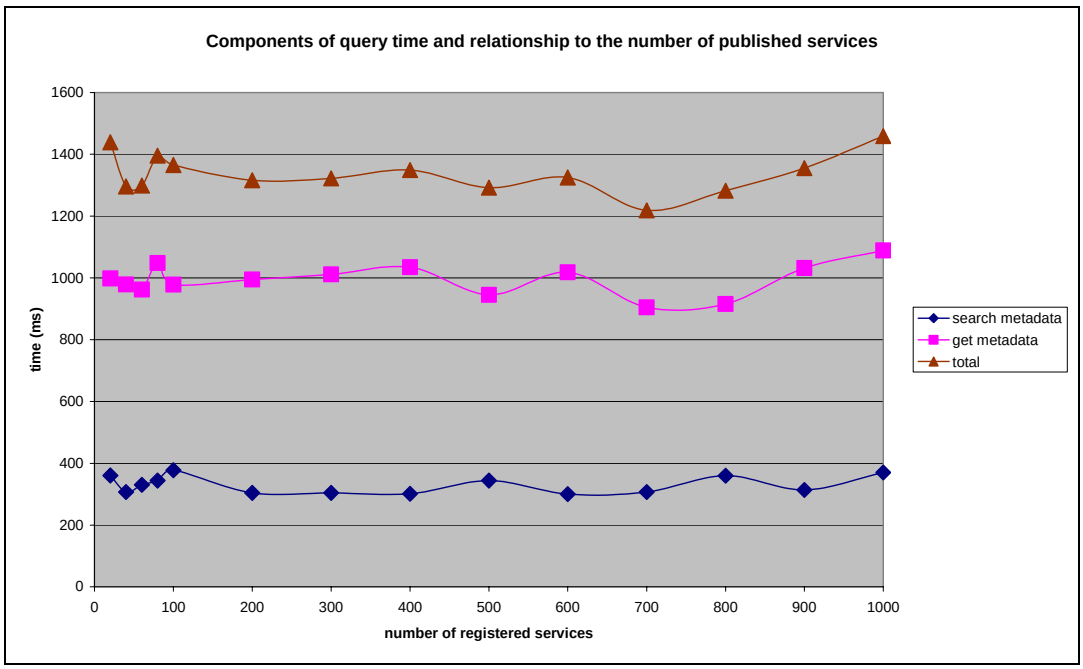

Figure 10. Query performance

In the results above, `total` represents the total time taken to execute the `Registry.getProcess()` method. `search metadata` is the observed time on the client to execute the synchronous call to the Grimoires server to execute the RDQL query. Similarly, the `get metadata` values are those observed on the client for the second network call to the server. This involves retrieving the metadata attachment that contains the OWL-WS description matching the initial query.

It can be seen from these results that there is no clear relationship between query time and the number of published services over this range (0 - 1000 registrations). There's a hint of performance beginning to scale linearly with number of published services starting at around

700 published services, but this is not seen when the number of published services is increased significantly (see later).

Again, however, the total times are very high. Furthermore, it is surprising that the time taken to execute the RDQL query is approximately 300ms whilst the time taken to retrieve the metadata attachment is around 1000ms. It might be expected that query time would be relatively more demanding of server resources, whilst the operation to retrieve a metadata attachment - a first class entity in the Grimoires data model - would be relatively quick. Further profiling within the Grimoires code base is required to identify the cause of this problem. However, initial review of the code reveals a likely cause. When retrieving a metadata attachment the current code issues numerous queries to the underlying triple store, to retrieve a tree of RDF triples, rooted at the node in the graph that represents the metadata attachment. Restructuring the persistence model and indexing of the metadata attachments and other Grimoires entities such that they can be retrieved atomically, will likely alleviate this problem and increase query performance significantly.

6.2.3 Comparison with WSRF-SG

Query of Grimoires was next considered at higher numbers of published services. The results are published below alongside results obtained previously for measurements of a prototype implementation of a WSRF-SG registry [10]. These results have been obtained under different test conditions and therefore are not directly comparable, but the overall relationship is still useful to see.

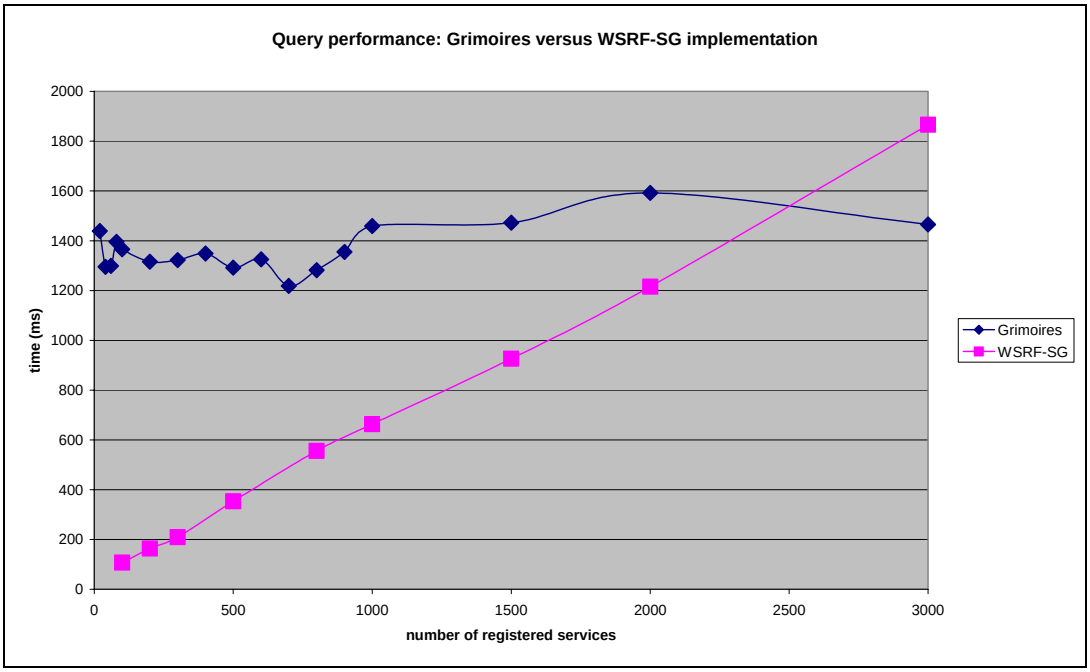

Figure 11. Query performance scaling compared to WSRF-SG

Query over the Grimoires registry to select and retrieve an OWL-WS `Process` takes a total time within 1200-1600ms. The lack of relationship between number of registered services and query time indicates that while queries over semantic descriptions are very complex, the Grimoires implementation does use indexing to ensure good scalability for large numbers of services. In contrast, it appears that the WSRF-SG registry may be configured for optimal update and query times increase linearly with number of published services.

6.2.4 Summary

The initial query values for Grimoires are very high and further profiling, within the server, is clearly required to determine possible optimisations. In these tests, XML serialisation and web service interactions likely add significant overhead. As determined above though, the `get metadata` component of the query contributes around 1000ms to the query time and this overhead could be almost entirely eliminated. This could be done by providing a simple server-side method that performs both semantic query and retrieval of the Grimoires entity in an atomic operation, thus reducing the number of client-server interactions from two to one call. This would eliminate approximately half the present network and web service stack overhead. Optimisation of the data structure and retrieval method for obtaining from the persistence layer the metadata attachment can also be performed relatively easily. As mentioned above, presently this involves numerous queries, one for each RDF triple and this could be redesigned such that a single query is used.

7 Summary of Conclusions

Semantic registry technology is required in NextGRID to support publication of OWL-WS descriptions of workflows, services and Grid applications; and to support query over complex semi-structured data that describe these entities and include information on non-functional properties, including those relevant to SLA.

A number of candidate registries have been reviewed and Grimoires has been identified as a good starting point. OWL-WS descriptions can be published in a Grimoires registry and it supports query with RDQL. Other registries considered, including vanilla UDDI and a WSRF-SG implementation, lack adequate query capability for OWL-WS and semi-structured data.

Functional and performance evaluation of Grimoires have identified some areas in need of improvement. Whilst RDQL is more powerful than traditional directory name/value pair searching, it is not able to capture queries that include disjunction of graph patterns. This means that it is not possible to express a query that matches two service descriptions that have different graph structures. It is expected that this and other limitations will impact registry clients in NextGRID. The SPARQL query language supersedes RDQL, and overcomes these limitations. It is proposed that enhancements are made to Grimoires to support SPARQL for semantic query. During testing it was found that there were some problems with publication and retrieval of metadata attachments. Metadata attachments are stored in a triple store within a single graph. This lack of structure can cause data integrity and performance problems when a client needs to retrieve an entire attachment. It's proposed that Grimoires internal data structure and persistence model could be improved such that appropriate structure is imposed. This will allow metadata attachments to be created, retrieved, updated and deleted without data integrity problems. Another issue is that a mechanism for adding custom ontologies and reasoners to Grimoires is required. Adding ontology and reasoning support will increase query power significantly and allow type hierarchies and other types of inference to be performed in the server instead of in clients.

Initial evaluation of Grimoires publication and query performance with varying numbers of published services has revealed that query and publication times are high. Further performance profiling at finer granularity within the Grimoires code is required to identify causes and solutions. Furthermore, more performance profiling is needed to assess scalability with varying load in terms of concurrent client requests, to assess how the registry performs under the dynamic environmental conditions expected in NextGRID.

The comparison of performance with the earlier WSRF-SG implementation in NextGRID is also revealing. The WSRF-SG code processes simple queries much faster than Grimoires processes semantic queries, as expected. However, the WSRF-SG code also scales poorly, and if there are too many services registered it even becomes slower than the semantic registry. It seems likely this is because the WSRF-SG software sacrifices query scalability for faster updates, which makes sense in soft-state registries that use frequent renewal or update notifications. This shows very clearly how different types of registries with different requirements may need completely different implementations. The existing WSRF-SG code would make an excellent "cache" for registering small numbers of pre-resolved services known to be of interest to a user that they may need to find and access with very low latency (essentially the case it was designed for). The same code would be far less effective as a "global" registry (because of scalability), or as a semantic registry (because it can not deal with workflows that do not have EPRs). It would be possible to adapt the existing WSRF-SG

code to make it better in these situations, but only by sacrificing the qualities that make it good as a service cache. The conclusion is that NextGRID should not try to produce one registry to suit all its needs, but should allow for different implementations, and where appropriate even different registry specifications, for global address registries, semantic functionality registries and service caches.

Overall, Grimoires as adapted and used in this experiment is a good starting point for a semantic registry suitable for use by the Grid VIM and more generally in NextGRID.

7.1 Answers to architectural questions

Based on the results of this experiment, we can provide the following answers to the questions addressed by this experiment, as discussed in Section 2.3.

Semantic service description

- 1) What semantic information is required to describe NextGRID services?
  - Functional properties (input and output data and types, application effect, etc), and non-functional properties (security requirements, business processes, etc), much of which can be described in terms of workflows.
- 2) What standards for semantic service description exist and are most appropriate for NextGRID?
  - RDF provides the basic semantic description standard, and OWL-S provides a standard way to use RDF to define workflows (hence much of the above service information). OWL-S appears well suited to the service description needs of NextGRID.
  - Further work should be carried out to investigate alternatives that seem useful in other areas of NextGRID.
- 3) What query functionality is required?
  - The RDQL language has been used in the experiment reported here, and was found to be insufficient for NextGRID purposes except in simple situations.
  - The SPARQL language has been investigated, and seems much more suitable, but could not be used in the current experiments. Further work should be carried out to evaluate this alternative in an updated service and workflow registry.

Service description

- 1) What further information needs to be in the description (beyond WSDL 1.1)?
  - Service dynamics must be published, and to do this requires a semantic workflow description, going well beyond WSDL 1.1.

Service discovery

- 56.5) Do we use OWL/RDF to describe the semantics?
  - This is possible using OWL-S, plus extensions defined for OWL-WS.
- 83) Is there a way to integrate process with dynamic policy description? How do we make this into an architectural component?
  - It is possible to describe process along with other attributes of a service using OWL-S and semantic queries. However, this experiment did not address how this information should be used by other new or existing components.

8 References

For more information, see:

1. The Next Generation Grid, Annex I: Description of Work, V1.0, EC IST Project 511563.
2. D.Snelling, M.Fisher, A.Basermann (eds), NextGRID Vision and Architecture White Paper.
3. NextGRID Project Output P5.3.2: OWL-WS implementation.
4. The NextGRID Architecture Straw Man. Available from NextGRID WP1.
5. Resource Description Framework (RDF). http://www.w3.org/RDF/
6. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/
7. Universal Discovery, Description and Integration v2. Available via http://www.oasis-open.org/specs/index.php#uddiv2
8. Web Services Service Group 1.2 Public Review Draft 02, 07 Oct'05, available from the OASIS WSRF TC.
9. Web Services Resource Properties 1.2 2 Public Review Draft 02, 06 Oct'05, available from the OASIS WSRF TC.
10. P.Hasselmeyer, NextGRID Project Output P5.2.1: Dynamic Distributed Registries and Protocols v1.0, 28 Sep 2005.
11. Feta semantic discovery. Published by the UK e-Science myGrid project, accessible via http://www.mygrid.org.uk.
12. RDQL - A Query Language for RDF. http://www.w3.org/Submission/RDQL/
13. Grid Registry with Metadata Oriented Interface: Robustness, Efficiency, Security. See http://twiki.grimoires.org/bin/view/Grimoires/.
14. The secure web services based Grid infrastructure, GRIA. http://www.gria.org
15. SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/