

Views on the View

Phillip Lord, Chris Wroe, Simon Miles, Nedim Alpedmir, Pinar Alper

May 4, 2006

Original document Date May 2004

1 Introduction

These notes are the summary and conclusions of a meeting between the listed authors, with additional contributions from Tom Oinn and Chris Greenhalgh. The meeting concerned the current requirements from WP-4 and the use cases on publishing and discovery and, in particular, understanding why the registry does not yet meet them and how we can improve the service so that it will. We also discussed the suggested way of defining, registering and discovering services specified by the *my*Grid information model, and its adequacy in meeting the use case requirements.

Broadly, the following points were made.

- The registry is in part based on requirements given by WP-4 for semantic service discovery, some of which have now been found to be invalid.
- WP-4 wishes to develop a small and discrete piece of software, focussed on user-centred semantic service discovery. This will enable us to finalize the data model and the queries that we wish to support discovery from within Taverna.
- WP-4 has no interest in replicating the registry's functionality and is aware that there are immediate requirements for this functionality, in particular multiple publishers, concurrency, and third party metadata.
- The registry implements several protocols, none of which are flexible enough to allow experimentation to determine the appropriate data model to support *my*Grid's requirements for semantic service discovery.
- The registry is not ontology-aware so, from WP-4's perspective, there is little value in registering the semantic metadata such that it is transparent to the registry. However, we are agreed that the idea of separation of the service registry and semantic service discovery remains valid.
- The prototype search technology and user interfaces being developed by WP-4, should serve as a good test-bed from which to enable appropriate future changes to the interface of the registry. Implementing the current information model over the current registry, which is (designed to be) fairly generic and may change based on WP-4's work, will not.

- From the past few months of working with the registry, WP-4 have found significant difficulties with the current implementation of the registry which should be addressed (JAX-RPC in particular).
- As a result of the discussions, we have developed a possible work plan which, we believe, will enable a convergence between the work currently being performed at Manchester and the registry implementation.

2 Requirements

In this section, we define the requirements that WP-4 have after attempting to use the registry for user-centred semantic service discovery, and then say why the current registry implementation does not meet them.

2.1 Services without WSDL Documents

We wish to register and describe many kinds of “service entitie”, which would include things such as workflows, SOAPLAB services, internally invocable Java classes, and so on. Most of these are not described (aqueately or at all!) by WSDL documents. This has become apparent since the original requirement gathering exercise for the view, as at that time, this multiplicity of invocable entities was not foreseen.

We want to publish these entities in the registry, but it does not seem sensible to require WSDL documents to be provided for these things. The first reason for this is that, in most cases, the generated WSDL document does not relate to the underlying invocation layer. Second, the WSDL may not be able to made remotely accessible by the advertiser so will not match the intended usage of a WSDL interface. ^{my}Grid requires the publishing of entities whose interface is not described in a WSDL document.

The registry currently requires ”service” interfaces to be advertised by submitting a reference to a WSDL document (or, as a more recent possibility, by submitting the text of the document).

2.2 User-Level Interface Descriptions

We require that the information model used for describing the interface of a ”service” be able to provide user-level descriptions of services, i.e. descriptions that will be useful in the user interface. For example, given the IBM BLAST service which we discussed in IEEE IS paper, we need a number of service operation invocations to do what is, from the user perspective, a single “operation”: to run a BLAST. The existence of several operations is clearly needed by the invocation layer, but it is not necessary for the discovery of this service and, further, is knowledge we wish to hide from the user, not present them with at discovery time.

WSDL describes the invocation layer and, even with metadata attached to entities within it, will not define the interface from the user perspective, which is what will be used in user-centred service discovery. As no other interface definition is allowed by the registry, this requirement is not fulfilled.

2.3 Updating Adverts

Due to the way in which semantic service descriptions are added, incrementally updating the service advert is the main way in which the registry is used for publishing from the discoveryview (semantic discovery) layer.

The registry client library instantiates service adverts as a network of objects when presented to a client and, as part of an advert may or may not exist (e.g. a bag may be empty), updating requires many checks on unrelated parts of the advert, which breaks attempts at encapsulation. Because of the complexity of the UDDI model this makes the update process very complex and error-prone.

None of the current convenience methods have the exact semantics required and these problems cannot really be solved by extra convenience methods at the present time. If many convenience methods are added, the client-side library could become as unwieldy as the fine-grained UDDI interface. Without knowing the particular updates that need to be made before experimentation, WP-4 cannot state the exact set of convenience update methods required.

2.4 Ontologies

Semantic service discovery must use ontologies to correctly resolve semantic queries.

The main thrust for discovery and ontologies during the project has been for a separation of concerns. The registry has been built without specific support for ontological reasoning, or discovery. We think that this is still the correct way forward.

However, it has a number of consequences. During the development of the discoveryview, we realised that the search capabilities provided by the view were not rich enough because they could not account for structure in the ontology. To get this functionality we would have had to do query rewriting in the discoveryview; even though the Jena backend of the view would have been capable of fulfilling the query directly if it had access to the ontology.

If we have to provide search technology within the discoveryview anyway, we seem to get relatively little benefit from structuring the semantic service descriptions so that they are transparent to the view, i.e. using the full functionality of the metadata attachment. We recognise that there is benefit to splitting up the description in this way in the situation where exact semantic matching is performed by, for example, a computational process at run-time, but there are currently no plans to implement this functionality withi *my*Grid.

This leaves us with two alternatives. First, we could introduce the ontology into the registry service, so that it is used during querying, breaking the separation of concerns. Alternatively, and perhaps more practically in the short term, we maintain this separation but return to storing the semantic service description as an opaque single blob of metadata. Other benefits that the registry provide, such as third-party assessments of services, are not affected by this latter choice.

2.5 Improved Performance

The service must be fast enough to demo. Registering a relatively large number of services in batch, and querying over these, is the expected use of the registry

within a demo.

Currently entering semantic service descriptions into the view can be very slow (particularly if using a persistent database backend where it can take up to an hour). Pinar has investigated this and most of the slowness lies in the JAX-RPC layer. This is exacerbated by the fine-grained UDDI methods, which are called in large groups by the client-side convenience methods so several Web Service calls are made for a single (convenience) operation. Service and WSDL registration is performed by making several calls over JAX-RPC which takes a considerable amount of time (4 mins 23 seconds for registering 29 services).

Querying over the registry provides better, but still slow, performance. However, we are currently not performing particularly complex queries, so this may become more of a problem as we move on. The actual execution of a query to find all registered services on the model performs as predicted (0-15 ms) on the server side. However when the corresponding API calls are timed at the client side the overhead added by JAX-RPC can be observed (the `findAllServices` operation takes 265 ms excluding `ProxyHelper` instantiation, 1157 ms including `ProxyHelper` instantiation).

When database persistence is used, all operations (both publish and find) are affected by this. As an example, a group of publish operations take an unacceptable amount of time when the model is persistently stored in RDB (55 minutes to register 29 services). However this should not be regarded as a performance problem of the registry implementation in particular: it is a problem with Jena in general. So the 50 minute difference between the in-memory and RDB model population processes is caused by Jena's handling of this difference.

As a development issue, the JAX-RPC interfaces also make for a very slow, and non-incremental, build process. This makes life hard, as it can lengthen the time it takes to determine whether any error is caused by a bug in the semantic discovery client or in the registry.

2.6 Reducing Complexity

If the registry interface is too complex, the clients themselves will become complex. This will increase the likelihood of errors in the client, so should be avoided. The internal complexity of the registry should not be complex, as it would then be likely to contain undetected bugs.

As stated in Section 2.3, the complexity of UDDI leads to increased complexity of clients and increased likelihood of errors. While conforming to standards is beneficial, UDDI does not provide any benefit for the semantic description of services. Few entities in the model are of any relevance to the services described in *myGrid*.

2.7 Experimentating to Finalise Information Model

The final requirement is the most important from the point of view of furthering *myGrid*, and the problems with meeting it are due to the requirements above not yet being met. We have some idea of the way in which services should be semantically described and discovered so that they are useful to users, and this is partly encoded in the draft *myGrid* information model, but need to finalise the interface and model by testing, i.e. giving users access to it in Taverna or

the *my*Grid portal. The user-centred semantic discovery envisaged fall into three categories: complex database-style queries (intended for development, testing, publication rather than users, task-based browsing by categorisation, and contextual, e.g. what could be validly inserted next in the workflow. This places an indirect requirement on the registry that it can be flexible enough to allow the metadata added and queries made to be rapidly varied as testing is conducted.

The registry implements a set of protocols which allow pre-defined queries. In addition arbitrary metadata can be added to individual entities in the UDDI and WSDL models and then queried over to discover entities of a given type, e.g. service, message part. However, there are problems with changing the semantic annotation quickly. First, the semantic description covers the whole service and, for the reasons given in Sections 2.1, 2.2 and 2.6 above, the currently annotated entities may not match that to which the metadata would best be attached. Second, changing the description annotated to an advert incrementally is a difficulty, as set out in Section 2.3. Finally, the poor performance described in Section 2.5 means that testing semantic discovery through the user interface is not rapid.

3 Experimental Semantic Search Component

To address the final requirement above, which is the most immediate for *my*Grid, WP-4 propose to create a test-bed for rapidly determining the descriptions and queries required for user-centred semantic service discovery. For ease of referral, they call this proposed component Feta. Feta would be based directly on a triple store, e.g. Jena, and provide the following things.

- A direct RDQL interface over the whole model.
- An interface providing canned queries as and when they are found to be useful through experimentation with the RDQL interface.
- A direct link to Pedro, so that updated service descriptions can be instantly published.
- User-centred operations (e.g. Web Service operation, workflow, SOAPLab service, Java class, sequence of Web Service calls etc.) being returned by the queries in a way that Taverna can interpret and display.
- Ontological reasoning in the processing of queries.
- User Interfaces integrated within Taverna which exercise all of the above.

Feta explicitly would not provide the following things.

- A registry service.
- A way for multiple clients to publish services.
- Any functionality specific to personalised views, e.g. third-party metadata, federation.

4 Conclusions

There are a number of concrete steps that we can take, to move forward from the current situation, which, hopefully, may allow us to converge the approaches being taken to service discovery in the future.

- Feta, as it stands, is a good place for rapidly prototyping the data model and queries that WP-4 wish to make against this data model. This will enable us to try this out in a “real user” setting, so that we can firm up this model.
- In the past, WP-4 have used the view as an opaque data store, where the service descriptions, at that time in DAML+OIL, were placed in the view, but not processed by it. We can incorporate the use of the view into the semantic service discovery process being trialled with Feta in the same way. This should provide us the benefits which the view offers: third party metadata, federation, personalisation and concurrent publishing by many sources, which Feta is not intended to replicate.
- The Feta backend will work over RDF, so we can publish this to the view, using its metadata interface, if we so choose. As a specific use case, this sort of functionality would be useful to select exact service matches (which are, therefore, mirrors) dynamically during workflow execution.
- The work done on Feta should enable us to test our ideas about the data model and the queries we require. It may make sense, following this process, to incorporate the interfaces from Feta directly as a new API for the view.