# Towards a protocol for the attachment of metadata to grid service descriptions and its use in semantic discovery

Simon Miles, Juri Papay, Terry Payne, Michael Luck and Luc Moreau
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
Tel: +44 23 8059 8309
Email: sm@ecs.soton.ac.uk

**Abstract**

Service discovery in large scale, open distributed systems is difficult because of the need to filter out services suitable to the task at hand from a potentially huge pool of possibilities. Semantic descriptions have been advocated as the key to expressive service discovery, but the most commonly used service descriptions and registry protocols do not support such descriptions in a general manner. In this paper, we present a protocol, its implementation and an API for registering semantic service descriptions and other task/user-specific metadata, and for discovering services according to these. Our approach is based on a mechanism for attaching structured and unstructured metadata, which we show to be applicable to multiple registry technologies. The result is an extremely flexible service registry that can be the basis of a sophisticated semantically-enhanced service discovery engine, an essential component of a Semantic Grid.

## 1 Introduction

Service discovery is a difficult task in large scale, open distributed systems such as the Grid and Web, due to the potentially large number of services advertised. Existing service discovery technology provides a partial solution, but there are several outstanding limitations related to the provision of extra information in a service advertisement. These include the automation of service discovery, allowing software tools to reduce the burden on users, discovery based on semantic descriptions of services, publishing of information about a service by others than the service provider, and publishing of enactable entities not normally considered to be services, such as workflows. In this paper, we have attempted to provide solutions to these four limitations, described below.

**Automated Service Discovery** In order to characterise their needs, users typically specify the complex requirements they have of a service, including demands on its functionality, quality of service, security, reputation and so on. Ideally, software tools can then match these requirements against the advertised capabilities of existing services, allowing discovery of appropriate services at any one time to be independent of the user. An example of such a tool, from the myGrid project [14], is of a *workflow enactment engine*. This takes a workflow description and invokes services in the order that the workflow defines. Services can be classified by their type and so the enactment engine can discover and choose between the available services as it enacts each activity of a workflow.

While schemas can be provided for organising the service capabilities in a structured form, information conforming to the schemas cannot be fully expressed in the constrained information models provided by existing service registry technologies such as UDDI (the de facto standard for Web Services registry), the OGSA Registry [8] or the Jini lookup service [1]. In these registries, information can be given as a textual description of a service, but cannot then be parsed and used programmatically by software tools.

**Semantic Discovery** More demandingly, user requirements may contain information that assumes an understanding of particular application domains and that is meant to affect the outcome of the discovery process. For example, a bioinformatician may search for services that process "expressed sequence tags" but may also be satisfied with those that operate on "nucleotide sequences". The knowledge that the latter biological concept is a more general form of the former is known by the user, but is unlikely to be expressible in either the query or the standard descriptions of useful services.

Given the plethora of services and sophistication of user requirements, many have advocated the use of automatic processing of service descriptions. In particular, to meet user requirements, it is proposed that the functional and non-functional characteristics of services should be expressed and reasoned over using semantic technologies [3, 12, 19].

**Third-Party Publishing** On the other hand, there may be information that could usefully be utilised in the discovery process, but is not or cannot be published by the service provider. For example, recommendations about which services to use may come from one member of an organisation, and could be used by their collaborators.

**Non-Standard Services** Finally, there are invocable or enactable entities other than services that need to be published. For example, *workflow scripts* describe how to compose services together to get a useful composite behaviour. They are location-independent descriptions of service use and should be publishable in the same registries as services since they provide comparable functionality, i.e. they are invocable, take inputs and return outputs (a similar equivalence is recognised explicitly in OWL-S [12]).

## 1.1 Paper Summary

In this paper, we report on our technical solutions to address the above limitations, specifically focusing on the following contributions.

- We have developed a protocol for attaching metadata to registered service descriptions, and for querying over service descriptions and their metadata; a specific kind of metadata that our protocol supports is semantic description.
- We have implemented a UDDI-compatible service registry that allows metadata to be attached to various parts of the service description, such as the services themselves, their operations, their inputs and their outputs (as specified in WSDL documents describing service interfaces).

- An extensible programmatic interface (API) provides clients with an easy way to access this information, whether it is held in a remote Web Service or locally.
- We demonstrate, with a common information model, that our attachment methodology applies to multiple discovery technologies in a single registry, including UDDI [16], OWL-S [11] and BioMOBY [17].
- We apply the metadata attachment protocol to allow service adverts to be augmented with semantic descriptions and for these to be used in semantic discovery.

These developments are part of the architecture developed by the myGrid project (`www.mygrid.org.uk`). They constitute a core component, which we use in practice to publish semantic service descriptions and reason over them. The reader interested in issues related to semantic reasoning over the service descriptions stored in our registry in myGrid is referred to [10], and in performance appraisals of our service to [13].

In this paper, we will first examine the limitations of existing approaches (Section 2) and then consider how to overcome these by allowing for annotation of service descriptions by a metadata attachment mechanism (Section 3). We then examine the protocol by which a client can register and query over metadata (Section 4), and in Section 5 we discuss how our services can help in overcoming some of the limitations mentioned above. Finally, we discuss the implications of our work and its current and future developments in Section 6.

## 2 Limitation of Existing Approaches

A few registry technologies and service capability languages already exist, but each has limitations indicating that they cannot tackle the problems expressed above. Here, we describe two technologies that are prominent in current Web Service-oriented architectures, UDDI and WSDL, and two that make some attempt to solve the problems above, OWL-S and BioMOBY, and explain why each does not provide a complete solution.

### 2.1 UDDI

The UDDI service registry (Universal Description, Discovery, and Integration) [16] has become the de-facto standard for service discovery in the Web Services community. Service queries are typically white pages-based or yellow pages-based: services are located using a description of their provider or a specific classification (taken from a published taxonomy) of the desired service type. In UDDI, service descriptions are composed from a limited set of high-level data constructs (*Business Entity*, *Business Service* etc.) which can include other constructs following a rigid schema. Some of these constructs, such as *tModels*, *Category Bags* and *Identifier Bags*, can be seen as metadata associated with the service description. However, while useful in a limited way, they are all very restrictive in their scope of description and their use in searching the registry. In particular, services are not the only entities that need to be classified. For instance, classifications can also be defined for individual operations or their argument types. However, it is not convenient to use searching mechanisms for services that are distinct from those for their argument types. Likewise, the fact that a *tModel*'s reference is to an external technical specification, such as a WSDL file describing a service interface, also implies that a different mechanism is required for reasoning over service interfaces. These are clear restrictions of the facilities offered for attaching metadata to entities in UDDI service descriptions.

In addition, UDDI provides no data structures to represent either the abstract or concrete details contained within a WSDL document, but only a standard way to express that a service implements a particular WSDL interface. A new proposal allows tModels to reference specific bindings and port types [5]. However, this extension still does not provide access to, or queries over, operations or messages, which would allow the discovery of services capable of specific operations.

## 2.2 WSDL

Some uses of the WSDL interface definition language itself present limitations too, as illustrated by Figure 1, which displays the interface of an existing bioinformatics service called BLAST. BLAST is a widely used analysis tool that establishes the similarity of a DNA sequence of unknown function to already annotated ones to help the biologist gain an understanding of its possible function. We use BLAST in a data-intensive bioinformatics Grid application, which aims to make an educated guess of the gene involved in a disease and to design an experiment to be realised in the laboratory in order to validate the guess [7]. The BLAST service is invoked, as part of a workflow enacted over the Grid, on a large number of sequences without user intervention. The biologist runs the workflow to discover as much information as possible about a candidate gene, through calls to BLAST as well as multiple other services, such as the MedLine database of medical articles and the Gene Ontology. The results are subsequently used to create useful visualisations of the gene and to inform the next steps of the lab experiment.

The interface of Figure 1 identifies a portType composed of one operation, which takes an input message comprising two message parts, `in0` and `in1`. These parts are required to be of type `string`, but the specification does not tell us what the meaning of these strings is supposed to be. In most cases, they are sequences, for which many formats are supported. This example was chosen because it precisely illustrates limitations of some existing service descriptions. While this interface specification could easily be refined by using an XSD complex type [4], it is unrealistic to assume that all services in an open Grid environment will always be described with the appropriate level of detail. Moreover, should it be so, we cannot expect all service providers always to use type definitions expressed with the terms of reference adopted by a given user.

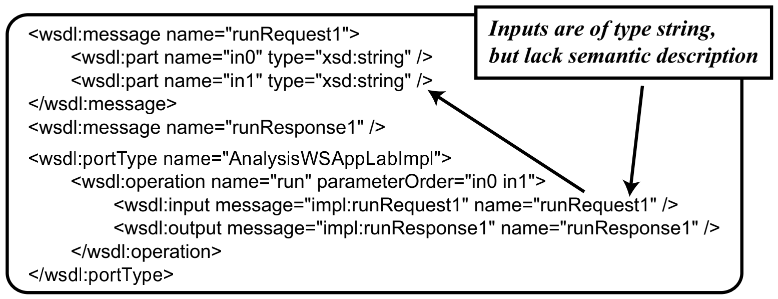

```
<wsdl:message name="runRequest1">
      <wsdl:part name="in0" type="xsd:string" />
      <wsdl:part name="in1" type="xsd:string" />
</wsdl:message>
<wsdl:message name="runResponse1" />
<wsdl:portType name="AnalysisWSAppLabImpl">
      <wsdl:operation name="run" parameterOrder="in0 in1">
            <wsdl:input message="impl:runRequest1" name="runRequest1" />
            <wsdl:output message="impl:runResponse1" name="runResponse1" />
      </wsdl:operation>
</wsdl:portType>
```

Figure 1: Basic Local Alignment Search Tool (BLAST) Interface Excerpt

Other problems relate to the rigid nature of both UDDI and WSDL, and the lack of metadata attachment capabilities, can be seen when looking at the uses to which they are put. A UDDI query typically returns a list of available services, from which a subset may conform to a known and/or informally agreed upon policy and thus can be invoked. Such approaches work well within small, closed communities, where a priori definitions of signatures and data formats can be defined. However, across open systems such as the Grid, no assumption can be made about how desired services are described, how to interact with them, and how to interpret their corresponding results. Additionally, service providers typically adopt different ways to model and present services, often because of the subtle differences in the service itself. This raises the problem of *semantic inter-operability*, which is the capability of computer systems to meaningfully operate in conjunction with one another, even though the languages or structures with which they are described may be different. *Semantic discovery* is the process of discovering services capable of semantic interoperability.

For example, the BLAST service discussed earlier is made available to clients in very different ways. A deployment of the service by IBM can perform in four steps the same function that a service deployed at the European Bioinformatics Institute performs in three steps, due to the differences in setting up sessions and in providing each necessary piece of data. Overall, however, they both take the same information and perform the same task and so can both semantically interoperate with a service providing DNA sequences. More details of this example and the significance of different types of metadata can be found in [18].

## 2.3 OWL-S

In an attempt to address the lack of semantic description in service advertisements, OWL-S provides a description of a service that includes formal definitions of how the service can be enacted (through its constituent sub-components, or *atomic processes*) to achieve a goal. A full OWL-S service description incorporates three component perspectives: an abstract description of the service from the AI planning view, which includes: inputs, outputs, preconditions, and effects of a service (the Service Profile); the workflow view of the service, which consists of a set of processes (analogous to WSDL operations) that are composed together with formally defined control constructs to achieve some complex goal (the Process Model); and the mapping of the invocable atomic components within this workflow to corresponding concrete WSDL operations (the Service Grounding). Of interest to this paper is OWL-S' capability to attach semantic annotations to service descriptions. First, by its ontological nature, the OWL-S ontology may be subclassed to provide new information about services such as the task performed by a service or the algorithm it relies upon, as discussed in [19]. As OWL-S is defined as an OWL ontology, it supports the arbitrary attachment of semantic information to the parameters of a service. Indeed, the argument types referred to by the profile input and output parameters are *semantically grounded*. Such semantic types are mapped to the syntactic type specified in the WSDL interface by the intermediary of the service grounding. We feel that such a mechanism is a step in the right direction, but it is convoluted (in particular, because the mapping from semantic to syntactic types involves the process model, which we did not discuss). It also has some limitations since profiles descriptions do not readily support multiple interpretations or definitions of semantically defined parameters. Finally, such semantic annotations are restricted to input and output parameters, but may not be applied, in a similar manner, to other elements of a WSDL interface specification.

## 2.4 BioMOBY

As a result of these considerations, we require a registry technology that combines semantic description of both input and output parameters as well as services with explicit matching of those semantic types to synactic formats. BioMOBY [17] is a service discovery architecture based on a view of a service as an atomic process or operation that takes a set of inputs and produces a set of outputs. The service, inputs and outputs can all be characterised by semantic types; inputs and outputs will also have syntactic types. So, for example, a service provider may register a BLAST service to take (semantically) nucleotide sequences (syntactically, simple strings) and perhaps (semantically) a BLAST e-value cutoff (syntactically, a real number), and produce a set of matching sequences and e-values. It is limited in that it does not support the UDDI protocol, so specialist clients have to be developed, and it cannot have general extensions of service descriptions, so, for example, the attachment of quality of service ratings or structured descriptions to service adverts is not supported.

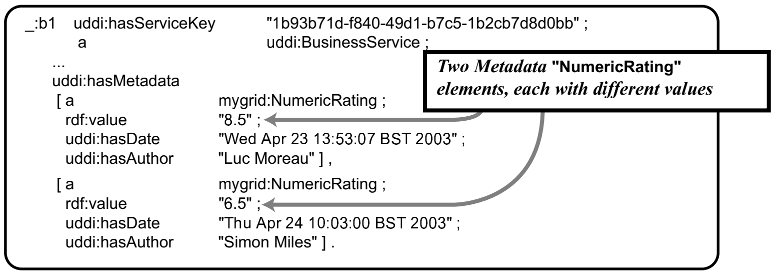
## 3 Extending Service Descriptions

Having discussed the limitations of existing technologies, we now focus on the capabilities of our service registry, which allows for extension of service descriptions by adding *metadata attachments*. A metadata attachment is an extra piece of data giving information about an existing entity in a service description, and is explicitly associated with that entity. Entities to which metadata can be attached include the service itself, an operation supported by the service, an input or output type of an operation invocable on the service. The metadata is attached by calls to the service registry after publishing, with reference to the entity to which the metadata should be attached. To establish that this attachment mechanism is generic, we have applied it to the service descriptions supported by UDDI, WSDL, OWL-S and BioMOBY, which we all encode with a common information model.

We have adopted RDF triples [15] to represent all descriptions of services. RDF (Resource Description Framework) is an XML data format for describing Web and Grid resources and the relations between them. Triples are simple expressions of the relations between resources, consisting of a subject, a relation (or property) and an object. All our triples are stored in a triple store, which is a database whose interface and implementation are specially designed to hold such triples. Specifically, we rely on the Jena implementation [9] of such a triple store.

To illustrate our general mechanism, we consider different kinds of metadata attachment, for which we found practical uses in our Grid application: *(i)* attaching ratings to services; *(ii)* attaching functionality profiles to services; and *(iii)* attaching semantic types to operation arguments. Ratings can provide users with assessments from experts on the value of a particular service; functionality profiles can be used to refine a search to exactly those services that are relevant; and semantic types allow clients to ask whether services are applicable to the type of data they have (so overcoming the limitations of WSDL described above). Our presentation is based on examples that were generated by dumping the contents of our service registry; it relies on the N3 format [2], which we have chosen for its readability. For example, in Figure 2, we show the representation of a service annotated by two numerical ratings, with different values, and provided by different authors at different times. The service is described by many pieces of information from the standard UDDI model such as its service key (the value to the right of `uddi:hasServiceKey`), and by two pieces of metadata attached to it (under `uddi:hasMetadata`). Each piece of metadata has a type (in this case, both are of type `mygrid:NumericRating`), a value (the rating itself) and two pieces of provenance information. The provenance information is the time and date at which the metadata was published and the name of the publisher. Such information is particularly useful in a registry allowing third parties to attach ratings because it can be used to identify the origin of an annotation.

In myGrid, we describe services by a service profile [19] specifying which particular type of process a service uses to perform its task (`uses_method`), which ap-



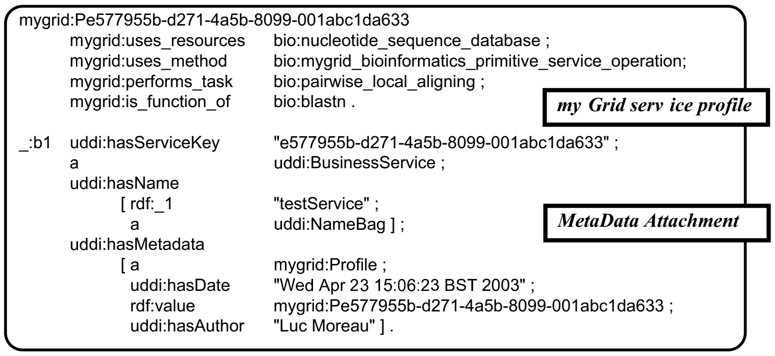
```
_:b1   uddi:hasServiceKey          "1b93b71d-f840-49d1-b7c5-1b2cb7d8d0bb" ;
        a                          uddi:BusinessService ;
    ...
    uddi:hasMetadata
     [ a                   mygrid:NumericRating ;
       rdf:value           "8.5" ;
       uddi:hasDate        "Wed Apr 23 13:53:07 BST 2003" ;
       uddi:hasAuthor      "Luc Moreau" ] ,
     [ a                   mygrid:NumericRating ;
       rdf:value           "6.5" ;
       uddi:hasDate        "Thu Apr 24 10:03:00 BST 2003" ;
       uddi:hasAuthor      "Simon Miles" ] .
```

Two Metadata "NumericRating" elements, each with different values

Figure 2: Rating Attachment (N3 Notation)

plication domain specific task they perform (`perform_task`), which resources they use (`uses_resources`) and what application they are wrapping (`is_function_of`). A relevant excerpt of the service registry contents is displayed in Figure 3, with `b1` denoting a service and `Pe577955b-d271-4a5b-8099-001abc1da633` the "myGrid profile". This is useful because it presents clients with information that is concise but matches their knowledge of the service's application domain (bioinformatics in this case), and can be reasoned about by other semantic technologies using expert knowledge of the application domain available on the Grid.

In this specific example, it should be noted that the objects of the different triples, `nucleotide_sequence_database`, `mygrid_bioinformatics_primitive_service_operation`, `pairwise_local_aligning`, and `blastn`, are all concepts of a published ontology of the bioinformatics domain [19], which is not directly held in the repository, but is used for semantic reasoning over that domain. Semantic reasoning may allow a discovery mechanism to, for example, return a service annotated with `blastn` in response to an `alignment` request, since the latter subsumes the former [10]. It should be noted that the semantic reasoning, including inference over the semantic annotations, is not performed by the registry itself but by independent mechanisms. This design decision was made to allow separation of concerns, different reasoning mechanisms over the same service descriptions and to prevent a detriment to performance in searching over non-semantic annotations.

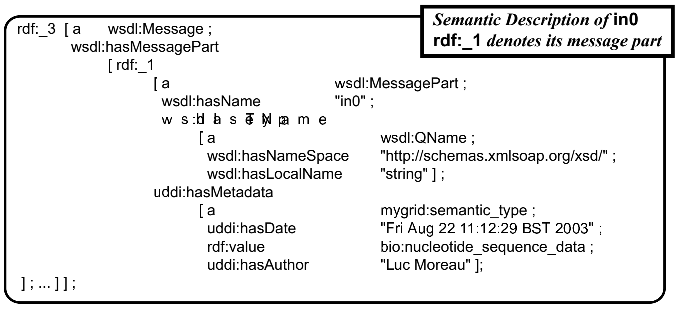


```
mygrid:Pe577955b-d271-4a5b-8099-001abc1da633
        mygrid:uses_resources     bio:nucleotide_sequence_database ;
        mygrid:uses_method        bio:mygrid_bioinformatics_primitive_service_operation;
        mygrid:performs_task      bio:pairwise_local_aligning ;
        mygrid:is_function_of     bio:blastn .

_:b1    uddi:hasServiceKey        "e577955b-d271-4a5b-8099-001abc1da633" ;
        a                         uddi:BusinessService ;
        uddi:hasName
              [ rdf:_1            "testService" ;
                a                 uddi:NameBag ] ;
        uddi:hasMetadata
              [ a                 mygrid:Profile ;
                uddi:hasDate      "Wed Apr 23 15:06:23 BST 2003" ;
                rdf:value         mygrid:Pe577955b-d271-4a5b-8099-001abc1da633 ;
                uddi:hasAuthor    "Luc Moreau" ] .
```

Figure 3: Attachment of a myGrid profile (N3 Notation)

In scientific application areas, the scientists often want to discover services that analyse the data they have obtained from experiments. As shown earlier, the description of input and output syntax, given for example by XSD definitions in WSDL, is not always adequate to determine whether the service is applicable for consuming or producing data of a particular semantic type. Figure 4 illustrates a semantic description of parameter `in0` declared in the interface of Figure 1. The node `rdf:_1` denotes the message part with name `in0`. It is given a metadata attachment, with value `bio:nucleotide_sequence_data`, which again refers to a term in the ontology of bioinformatics concepts [19].



***Semantic Description of* in0**
**rdf:_1** ***denotes its message part***

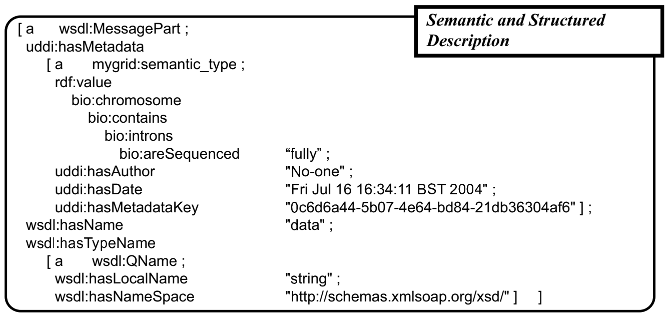
```
rdf:_3  [ a      wsdl:Message ;
         wsdl:hasMessagePart
               [ rdf:_1
                      [ a                           wsdl:MessagePart ;
                        wsdl:hasName                "in0" ;
                        wsdl:hasTypeName
                             [ a                           wsdl:QName ;
                               wsdl:hasNameSpace           "http://schemas.xmlsoap.org/xsd/" ;
                               wsdl:hasLocalName           "string" ] ;
                      uddi:hasMetadata
                             [ a                           mygrid:semantic_type ;
                               uddi:hasDate                "Fri Aug 22 11:12:29 BST 2003" ;
                               rdf:value                   bio:nucleotide_sequence_data ;
                               uddi:hasAuthor              "Luc Moreau" ];
 ] ; ... ] ] ;
```

Figure 4: Attachment of Semantic Types to Arguments (N3 Notation)

The benefits of such annotations are far reaching. Indeed, any service operating over strings could be composed with BLAST, and would produce a syntactically correct composition. However, these services would not inter-operate semantically since BLAST expects bioinformatics sequences and not arbitrary strings. By using the annotation mechanism, we make that information explicit, and hence promote semantic inter-operability of services.

Sometimes, however, the information needs to be more specific than reference to a single concept. Our protocol allows structured metadata to be attached to parts of the service advert, and then for services to be discovered using that metadata. In a real use case, a bioinformatician occasionally checks whether any updated or new chromosomes are available that they could analyse. Due to the nature of the experiment, they only care about those chromosomes that contain *introns*, which includes those from most organisms of at least the complexity of yeast and excludes most microbes such as viruses. This means that, at the start of their experiment, they wish to discover data with semantic type 'chromosome', but with the restriction 'contains introns'. A database file accessible over FTP can be advertised as the operation of a service that outputs, on request, chromosome data of a particular type. Those that return chromosome data that contains introns can be advertised as such, either by the service providers or, perhaps more likely, by a process that checks each file to try to determine whether each chromosome file contains introns and annotating the advert in the registry as a third-party. The bioinformatician is also interested in whether the introns are fully sequenced or not. If there are large gaps in the intron sequences, then their experiments are less meaningful, so ideally they would like chromosomes with fully sequenced chromosomes. This would lead to the description 'operation with output `data` of semantic type chromosome containing introns that are fully sequenced'. An RDF dump of the part of the registry containing these statements can be seen in Figure 5.

Multiple services could potentially have operations with the same semantic description described above despite being disparate and provided by independent parties, this mechanism provides a way to make services *semantically interoperable*. For example, in creating a workflow to perform the above experiment, services with different WSDL-defined interfaces could be discovered and included on the basis of their semantic description. However, other difficulties arise when trying to allow communication between different services, even when the semantics align, and we discuss many of these in [18].



Semantic and Structured Description

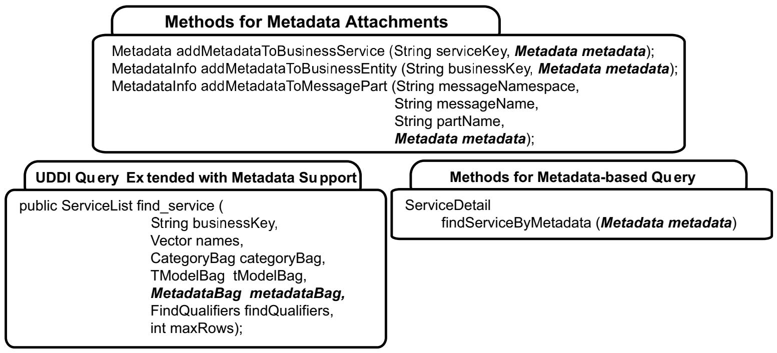
```
[ a     wsdl:MessagePart ;
  uddi:hasMetadata
     [ a      mygrid:semantic_type ;
       rdf:value
          bio:chromosome
             bio:contains
                bio:introns
                   bio:areSequenced        "fully" ;
       uddi:hasAuthor                      "No-one" ;
       uddi:hasDate                        "Fri Jul 16 16:34:11 BST 2004" ;
       uddi:hasMetadataKey                 "0c6d6a44-5b07-4e64-bd84-21db36304af6" ] ;
  wsdl:hasName                             "data" ;
  wsdl:hasTypeName
     [ a      wsdl:QName ;
       wsdl:hasLocalName                   "string" ;
       wsdl:hasNameSpace                   "http://schemas.xmlsoap.org/xsd/" ]    ]
```

Figure 5: Attaching Structured Information (N3 Notation)

## 4 Service Registry Protocol and Implementation

**Protocol** The protocol to publish metadata and discover services according to metadata was designed in a similar style to the UDDI protocol, so that UDDI clients could easily be extended to support such features. It is not possible to present the complete protocol in this paper. Instead, we refer the reader to the full set of commented interfaces, which can be accessed from `www.ecs.soton.ac.uk/~sm/myGrid /Views/`. As an illustration, Figure 6 shows some of the methods that allow the attachment of metadata, respectively to a business service, to a business entity, to an operation and to a message part. All these methods not only attach some metadata to the respective entity, but also add the aforementioned provenance information such as author and date of creation. The associated metadata can be structured or unstructured. Symmetrically, services can be discovered by using a metadata filtering mechanism. An example of metadata-based search method appears in Figure 6. Given some metadata, the `findServiceByMetadata` function returns the list of services that are annotated with such a metadata.

We support semantic discovery by allowing the metadata structure to contain terms of an ontology, which triggers ontological-based reasoning, the description of which is beyond the scope of this paper, but more details can be found in [10].



Methods for Metadata Attachments

```
Metadata addMetadataToBusinessService (String serviceKey, Metadata metadata);
MetadataInfo addMetadataToBusinessEntity (String businessKey, Metadata metadata);
MetadataInfo addMetadataToMessagePart (String messageNamespace,
                                       String messageName,
                                       String partName,
                                       Metadata metadata);
```

UDDI Query Extended with Metadata Support

```
public ServiceList find_service (
                  String businessKey,
                  Vector names,
                  CategoryBag categoryBag,
                  TModelBag  tModelBag,
                  MetadataBag  metadataBag,
                  FindQualifiers findQualifiers,
                  int maxRows);
```

Methods for Metadata-based Query

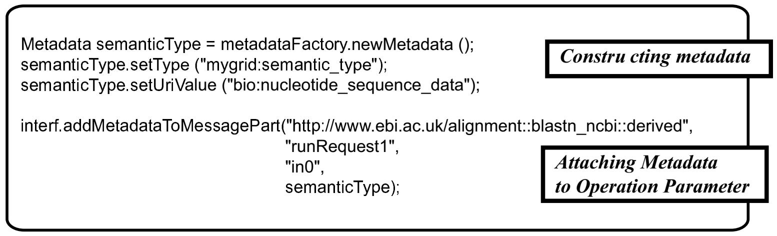
```
ServiceDetail
      findServiceByMetadata (Metadata metadata)
```

Figure 6: Metadata Attachment Methods

The benefit of our approach is the ability to extend some existing interfaces in an incremental manner, so as to facilitate an easier transition to semantic discovery for existing clients. For instance, we have extended the UDDI `find_service` method to support queries over metadata that would have been attached to published services. In the method specification of Figure 6, `metadataBag`, a new criterion for filtering services is introduced, which contains a set of metadata that a service must satisfy.

**API** We also provide a client-side library, or API, that makes it easy to use the protocol to communicate with the registry. Figure 7 illustrates how our API can be used to attach a semantic description to the first input of the `runRequest1` operation of Figure 1. The arguments of the `addMetadataToMessagePart` are the namespace of the service interface (an excerpt of which appears in Figure 1), the operation name (`runRequest1`), the parameter name (`in0`), and an object of type `Metadata`, whose type and values have been initialised to `semantic_type` and `nucleotide_sequence_data` respectively.

Several interfaces currently provide access to our general information model. Some of them preserve compatibility with the existing UDDI standard, and ensure inter-operability within the Web Services community. In Figure 7, we show an augmented UDDI `find_service` method, through which a client can discover services on the basis of its metadata annotation as well as its name, UDDI category, etc. A separate method, `findServiceByMetadata`, allows services to be discovered on the basis of their metadata alone, e.g. the semantic service profile described in Section 3.



```
Metadata semanticType = metadataFactory.newMetadata ();
semanticType.setType ("mygrid:semantic_type");
semanticType.setUriValue ("bio:nucleotide_sequence_data");

interf.addMetadataToMessagePart("http://www.ebi.ac.uk/alignment::blastn_ncbi::derived",
                                "runRequest1",
                                "in0",
                                semanticType);
```

Figure 7: Registering a parameter semantic type

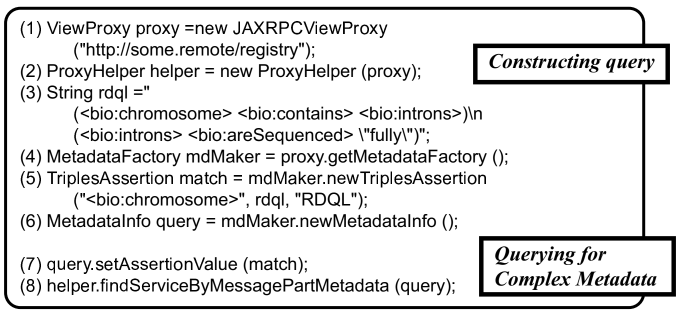
We show in Figure 8 the whole Java code to query for services to fulfill the structured metadata example in Section 3, where the bioinformatician requires services that output chromosome data containing fully sequenced introns. It uses the RDQL [9] language to define the metadata query. Working through the lines (numbered in the figure), we explain the purpose of each.

- First, in line 1, the querying client connects to the registry by creating a proxy object, which itself uses JAXRPC to forward all requests to the registry. In line 2, a ProxyHelper object is created: ProxyHelper provides a set of convenience methods to the client, hiding some of the complexity of the UDDI protocol.

- Lines 3 to 7 define the query that is used to find services that produce the required output type. The query itself, is expressed in the RDQL language on line 3, and consists of two connected statements that should be matched in the registry: a chromosome that contains introns, where those introns are sequenced fully. Lines 4, 5, 6 and 7 create the appropriate data structure used to submit the query to the registry. The term "bio:chromosome" on line 5 states that the chromosome is the root of the structured metadata.

- Finally, we make the call to the registry using a convenience method findServiceByMessagePartMetadata, which finds all services that have an input or output WSDL message part with the given metadata attached. This calls the registry and returns the matching set of service descriptions.

With inferencing based on ontologies, instead of exact matching, we could potentially find more applicable services. For example, if a service is advertised as returning human chromosomes (`bio:humanChromosome`), and an ontology states that



```
ViewProxy proxy =new JAXRPCViewProxy
        ("http://some.remote/registry");
ProxyHelper helper = new ProxyHelper (proxy);
String rdql ="
        (<bio:chromosome> <bio:contains> <bio:introns>)\n
        (<bio:introns> <bio:areSequenced> \"fully\")";
MetadataFactory mdMaker = proxy.getMetadataFactory ();
TriplesAssertion match = mdMaker.newTriplesAssertion
        ("<bio:chromosome>", rdql, "RDQL");
MetadataInfo query = mdMaker.newMetadataInfo ();

query.setAssertionValue (match);
helper.findServiceByMessagePartMetadata (query);
```

Figure 8: Querying for services outputting fully sequenced chromosomes

`bio:humanChromosome` is sub-concept of `bio:chromosome` then the query shown in Figure 8 would discover the service. Without inference, the service would not be found as `bio:chromosome` is not exactly the same term as `bio:humanChromosome`. In myGrid, we separate inference from discovery, to separate concerns and allow for greater performance where inference is not required. Our reasoning engine, performs semantic discovery by inferring from OWL-based ontologies and semantic service descriptions drawn from the registry.

Other registry interfaces, such as the interface to the triple store, directly expose the information model, and offer a powerful and radically different way of discovering services through the RDQL interface [9]. While such functionality is very useful, its radically different nature does not offer a smooth transition for client implementors wishing to adopt semantic discovery.

**Implementation** We have implemented the protocol in a registry, within which all service descriptions and metadata attachment are expressed as RDF triples and stored in the Jena triple store. We have designed and implemented a set of interfaces to the Jena triple store in order to offer a service registry functionality, for which Figure 9 depicts excerpts of class and collaboration diagrams. We see that a service registry implements a series of factory methods to create instances of interfaces to the triple store, which is passed as an argument to the factory methods and is itself created by a store factory. Different implementations of a store may exist, in memory or in a relational database [9].

There are four service registry interfaces shown: `PublishUDDI` and `InquiryUDDI` allow publishing and discovery of services using the UDDI protocol; `InquiryMetadata` provides searching based on the metadata attached to service description entities; and `WSDL` contains methods for publishing WSDL files to which metadata can be attached.

## 5 Discussion

We can now return to the initial Grid service discovery requirements presented in the introduction, and show how the work presented above addresses each.

**Automated Service Discovery** Using our service registry, metadata following client-defined schemas can be attached to service descriptions and then programmatically used in discovery. Metadata can be simple strings and URIs referring to other resources, such as ontological concepts. It can also be structured by giving typed relations between pieces of metadata using triples. This can be attached to service descriptions using the methods and technologies discussed in Section 4.

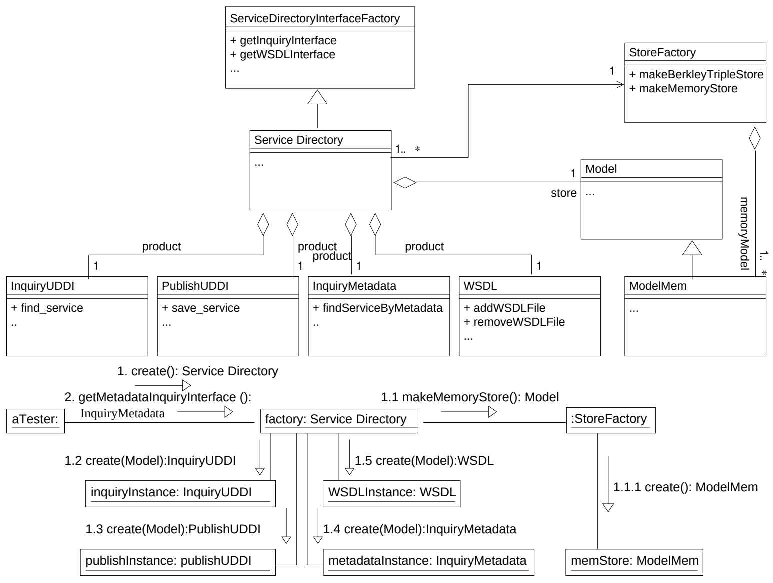


Figure 9: Class and Collaboration Diagram

**Semantic Discovery** Users can present queries that refer to terms in the domain in which they have expertise. Using the semantic terms that can be attached to service descriptions through our technology, along with the expert knowledge encoded in relevant ontologies and semantic reasoning, discovery of services will be widened or constrained based on the experience already built up by other knowledgeable professionals in the field.

**Third-Party Publishing** Recommendations regarding services, and opinions on their suitability and quality from users, can be attached as metadata to service descriptions and used in discovery. The provenance information shown in the above figures, giving the author and date of annotation for each piece of metadata, helps to distinguish metadata provided by multiple third parties. Clearly, this information may not be appropriate for storing in the public registries in which services are published. We are currently working on a factory to create 'views' over existing service registries, where a view is a personal copy of the contents of one or more other registries.

**Non-Standard Services** Workflow scripts and other location-independent processes can be published and discovered in our registry. Because they can be discovered and executed using the appropriate tools, they are directly comparable to service invocation. In terms of interfaces, workflows and parameterised queries both take inputs and provide outputs, each of which may be semantically annotated to enhance discovery.

## 6 Conclusion

In this paper, we have presented a protocol to publish semantic descriptions about services in order to promote semantic inter-operability. Our approach relies on a metadata attachment mechanism, capable of attaching metadata to any entity within a service description. Such metadata need not be published by service providers but can be published by third-party users. Our design extends the standard UDDI interface to provide semantic capabilities, thereby offering a smooth transition to semantic discovery for UDDI clients, and we have used these facilities to register service descriptions as specified by the myGrid ontology [19]. Future work will focus on providing service descriptions to WS-Resources (Web Services following the specification given by the WS-Resource Framework [6]). This has extra demands on top of those for Web Services due to, for example, WS-Resources having lifetimes controlled by WS-Resource Factories and a dynamic state. WS-Resource factories, in particular, require extra information to be stored in a service registry to allow clients to identify them as such and be informed as to how to use them to create the Grid Services they need.

## 7 Acknowledgement

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01). We acknowledge Keith Decker, Klaus-Peter Zauner, Carole Goble, Phillip Lord and Chris Wroe for their contributions to discussion on the work presented in this paper.

## References

[1] Arnod, O'Sullivand, Scheifler, Waldo, and Wollrath. *The Jini Specification.* Sun Microsystems, 1999.

[2] Tim Berners-Lee. Notation 3. http://www.w3.org/DesignIssues/Notation3, 1998.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[4] Paul V. Biron and Ashok Malhotra. Xml schema part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/, May 2001.

[5] John Colgrave and Karsten Januszewski. Using wsdl in a uddi registry (version 2.0). http://www.oasis-open.org/committees/uddi-spec/doc/draft/uddi-spec-tc-tn-wsdl-20030319-wd.htm, 2003.

[6] Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The ws-resource framework. http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf, 2004.

[7] R. Stevens et al. Performing *in silico* Experiments on the Grid: A Users Perspective. In S. Cox, editor, *Proceedings of the UK OST e-Science Second All Hands Meeting 2003*, pages 43–50, Nottingham, UK, 2003.

[8] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The Physiology of the Grid — An Open Grid Services Architecture for Distributed Systems Integration. Technical report, Argonne National Laboratory, 2002.

[9] Jena semantic web toolkit. `http://www.hpl.hp.com/semweb/jena.htm`.

[10] Phillip Lord, Chris Wroe, Robert Stevens, Carole Goble, Simon Miles, Luc Moreau, Keith Decker, Terry Payne, and Juri Papay. Semantic and Personalised Service Discovery. In W. K. Cheung and Y. Ye, editors, *WI/IAT 2003 Workshop on Knowledge Grid and Grid Intelligence*, pages 100–107, Halifax, Canada, October 2003. Department of Mathematics and Computing Science, Saint Marys University, Halifax, Nova Scotia, Canada.

[11] David Martin, Mark Burstein, Grit Denker, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Evren Sirin, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Owl-s 1.0 release. http://www.daml.org/services/owl-s/1.0/, 2004.

[12] David Martin, Mark Burstein, Ora Lassila, Massimo Paolucci, Terry Payne, and Sheila McIlraith. Describing web services using owl-s and wsdl. http://www.daml.org/services/owl-s/1.0/owl-s-wsdl.html, 2004.

[13] Simon Miles, Juri Papay, Vijay Dialani, Michael Luck, Keith Decker, Terry Payne, and Luc Moreau. Personalised grid service discovery. *IEE Proceedings Software: Special Issue on Performance Engineering*, 150(4):252–256, August 2003.

[14] myGrid - directly supporting the e-scientist. http://www.mygrid.org.uk/, 2001.

[15] Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2001.

[16] Universal Description, Discovery and Integration of Business of the Web. `www.uddi.org`, 2001.

[17] MD Wilkinson and M. Links. Biomoby: an open-source biological web services proposal. *Briefings In Bioinformatics*, 4(3), 2002.

[18] Chris Wroe, Carole Goble, Mark Greenwood, Phillip Lord, Simon Miles, Juri Papay, Terry Payne, and Luc Moreau. Automating experiments using semantic data on a bioinformatics grid. *IEEE Intelligent Systems*, 19(1):48–55, January 2004.

[19] Chris Wroe, Robert Stevens, Carole Goble, Angus Roberts, and Mark Greenwood. A suite of daml+oil ontologies to describe bioinformatics web services and data. *International Journal of Cooperative Information Systems*, 2003.