An Integrative Approach for Attaching Semantic Annotations to Service Descriptions

Luc Moreau, Juri Papay, Simon Miles, Terry Payne, Keith Decker

Department of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ 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 description and registry protocols do not support such descriptions in a general manner. In this paper we present an approach (and implementation) of service registry and discovery that uses an RDF triple store to integrates the major service descriptions (WSDL, UDDI, and DAML-S) and other, task- or even user-specific metadata that can be used for service discovery. Our approach allows us to present multiple interfaces (compatible with all the above approaches and others) so that services registered through one interface are available to be found via the other interfaces. Furthermore, we provide ways of extending these interfaces to attach other structured metadata, and search across it and all of the structures in the component models. The result is an extremely flexible service registry that can be used as-is with existing standards, or as the basis of a far more sophisticated semantically-enhanced service discovery engine.

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. In order to filter out the most suitable services for the task at hand, many have advocated the use of semantic descriptions that qualify functional and non-functional characteristics of services in a manner that is amenable to automatic processing [1, 4, 15].

Semantic discovery is the process of discovering services capable of interoperability, even though the languages or structures with which they are described may be different. Typically, a semantic discovery process relies on *semantic annotations*, containing high-level abstract descriptions of service requirements and behaviour. In this paper, our work focuses on the means to register and discover such semantic annotations.

Current standards in the Web Services and Grid communities do not directly support semantic discovery of services [11]. On the other hand, UDDI and WSDL

have the tremendous advantage of being standards agreed by the community; their existence is therefore essential to promote inter-operability with components such as workflow enactment engines based on [3].

An essential element in semantic discovery is to allow users to augment service descriptions with additional information, i.e. *metadata*. Providers may adopt various ways of describing their services, access polices, contract negotiation details etc. However, many resource consumers also impose their own selection policies on the services they prefer to utilise, such as provenance, derived quality of service, reputation metrics etc. Furthermore, it is useful to add such *metadata* not only to service descriptions, but also to any other concept that may influence the discovery process, e.g. supported operations, semantic types of arguments, businesses, users. Such metadata may be structured according to published ontologies, so that it can be interpreted unambiguously by multiple users, especially in the case of a public registry; alternatively, such metadata may also be raw and unstructured, in the case of a personal registry used by a single user. Relevant initiatives in that area include the following: DAML-S [4] has some support for attaching semantic annotations to services and their arguments; BioMOBY [14] allows semantic descriptions of operations and arguments; our previous work already introduced the idea of metadata attachment [9].

In summary, current standards are not capable of semantic service descriptions, but promising initiatives allow semantic attachments. Against this backgound, we believe that an information model, i.e. an ontology, unifying not only UDDI and WSDL descriptions, but also general metadata attachment [9], and DAML-S and BioMOBY -style semantic annotations, would provide us with uniform way of querying and navigating service information. We see the use of RDF triples [12] (subject, predicate, object) as the means to represent all the information in a uniform manner. This information will be stored in a triple store, which can be queried uniformly through the use of a query language such as RDQL [7]. While our integrative information model is suitable for expressing semantic descriptions, it also critical to offer programmatic interfaces that would allow both publishers and third-party users to register their semantic information. Therefore, on top of this triple store, we have implemented standard and emerging interfaces such as UDDI, DAML-S, and a BioMOBY-like functionality.

This work should be seen as a building block of the myGrid (`www.mygrid.org.uk`) architecture for semantic service discovery. The functionality we are discussing here allows the attachment of semantic annotations to services descriptions; such semantic descriptions can be retrieved, and used for reasoning by a *Semantic FindService* component, whose description and interaction with the current component are discussed in a companion paper [cite].

The specific contributions of this paper are the following:

1. We have defined a unifying ontology that links the service descriptions supported by UDDI, WSDL, DAML-S, and BioMOBY.
2. We have adopted RDF triples as underlying representation of the information, we use a triple store for storing such information, and we support the RDQL query language for uniform navigation of the information.

3. Over this stored information, we have specified multiple interfaces offering compatibility with multiple standards or emerging approaches (UDDI, WSDL, BioMOBY, DAML-S). Services registered through one interface are available to be found via the other interfaces.
4. We also provide extended interfaces for these standards and emerging approaches that allow the attachment and query of both simple and structured metadata. As an illustration, we make use of these interfaces to register and find services described according to the myGrid ontology [15]
5. Our design is backed up by an implementation of a service directory capable of semantic annotations, which can be found at `http://www.mygrid.ecs.soton.ac.uk/service-directory/javadoc`.

This paper is organised as follows. In Section 2, we review existing approaches and discuss their limitations, hereby building up motivation for a unifying information model, which we present in Section 3. We then discuss the multiple interfaces that can be supported for this information model, and discuss briefly our implementation in Section 4. A benefit of our approach is that it underlines the differences between various "notions of services", which we discuss in Section 5; we also highlight the benefit of our approach by showing how existing standard interfaces can be enchriched by semantic-oriented facilities, offering an transitional approach to semantic service directory.

2 Background

The Web Services community has adopted an open standards approach for locating, describing, choreographing, and enacting heterogeneous services. SOAP [17] provides a transport mechanism to shuttle messages between services that are functionally described by WSDL [16]. Larger services may utilise workflow based languages (such as WSFL [8] or BPEL4WS [3]) to describe the coordination of serveral services. These services are registered with a UDDI service directory [13]. Service queries are typically white or yellow pages based: they are located based on a description of their provider or a specific classification (taken from a published taxonomy) of the desired service type. This 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, assumptions cannot be made about how desired services are described; how to interact with them, and how to interpret their corresponding results. Many of the existing systems assume some form of adherence to a standard or agreed model. For example, printer services within a Jini environment [10] are expected to share the same signature and provide a core subset of functionality so that consumers looking for "printer services" will be able to interoperate. However, service providers typically vary in the way they 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 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 inter-operability.

Current standards in the Web Services and Grid communities do not support semantic discovery of services [11]. The UDDI (Universal Description, Discovery, and Integration) standard supports a construct called TModel which essentially serves two purposes: it can serve as a namespace for a taxonomy or as a proxy for a technical specification that lives outside the registry [6]. We believe that such a TModel construct has some intrisinc limitations. While there is no doubt that service classifications are useful, services are not the only entities to be classified. Classifications can also be defined for individual operations or their argument types. Why should we use searching mechanisms for services that are distinct from those for their argument types? Likewise, a TModel's reference 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.

DAML-S [4] is a set of ontologies designed for describing, choreographing and invoking services and workflows within open, distributed systems. Built upon the DARPA Agent Markup Language (DAML) [], it exploits the definition of concepts defined and distributed across the Semantic Web [1], and the ability to reason across these concepts with respect to referential ontologies so as to identify and interpret service descriptions at the semantic level. DAML-S provides four high level ontologies, which can be employed, or subclassed, to facilitate the modeling of service descriptions. The *Service* model represents the service itself, and presents three different views on the service; the *Profile* model, which describes what the service does (in terms of a capability description); the *Process* model, which describe how the service works (in term of a process workflow), and a *Grounding* model, which maps the process workflow to a WSDL description of the service.

Of interest to this paper is DAML-S' capability to attach semantic annotations to service descriptions. First, by its ontological nature, the DAML-S ontology may be subclassed to provide new information about services such as, e.g. the task performed by a service or the algorithm it relies upon, as discussed in [15]. We will refer to this kind of semantic description as *ontology-design time* since description types, e.g. the task performed by a service, are decided at ontology design time. Furthermore, such semantic descriptions apply to classes of services and not service instances.

DAML-S provides an alternate mechanism that allows service publishers to attach semantic information to the parameters of a service. Indeed, the argument types referred to by the profile input and output parameters are *semantic*. 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 it only supports semantic

annotations provided by the publisher, and not by third party annotators; a profile only supports one semantic description per parameter and does not allow multiple interpretations. 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.

In the bioinformatics community, BioMOBY [14] is highly visible project whose aim is to explore methodologies for representing, distributing and discovery biological data. BioMOBY Central is the registry through which services can be published and discovered. BioMOBY Central has some support for semantic description, which we now outline. BioMOBY services are seen as atomic operations; for each operation, both the syntactic and semantic types of the inputs and outputs have to be specified. Symmetrically, a service can be discovered according to the semantic types of their inputs and outputs. In fact, the BioMOBY way of registering information has strong similarities with DAML-S: the capabilities are a subset of the DAML-S-capabilities but the mechanism is lighter weight (there is no process description, since each service is an atomic operation, and there is no grounding as the WSDL description can be generated on-the-fly). Like DAML-S, such semantic descriptions essentially remain limited to inputs and outputs; for each of them, only a publisher's annotation is supported, without any support for third party annotations.

Therefore, we favour the use of an integrative information model capable of the kind of semantic annotations supported by DAML-S and BioMOBY, but still preserving the UDDI and WSDL standards. Our proposed approach differs radically from the one advocated in [11]: they propose a mechanism by which, whenever a DAML-S service is registered, an equivalent service is automatically registered in a UDDI registry, therefore allowing such a service to be discovered by the regular UDDI protocol. This approach is not satisfactory for two reasons: first, it does not allow a service registered by the UDDI interface to be discovered by the DAML-S one; second, it does not have a single information repository including both UDDI and DAML-S information that can be navigated uniformly. We address these two deficiencies in our design.

3 Integrating Different Services Description Models

In this section, we present the information model integrating different approaches for service descriptions. All our ontologies are made available from `http://www.mygrid.ecs.soton.ac.uk/service-directory/`. We have chosen three models as the basis of our integration; UDDI, WSDL, and DAML-S. As we have just dicussed, each brings with it different strengths and weaknesses, and also two different, but complementary conceptualizations of what a service is, namely, *service-as-endpoint* and *service-as-goal-achieving-process*.

For specifications driven mostly by the traditional distributed computing community (UDDI, WSDL) "service" tends to indicate a physical computing *entity or entities* that present some well-specified interface at specific physical *endpoints*. For specifications driven by the software agent, or more general AI

community (DAML-S, BioMOBY), "service" tends to indicate a *process by which one may achieve a goal*. These two viewpoints have significant overlap—an extremely common case in specification examples and in real implementations is one where a physical computing entity presents a single well-specified interface which in turn enacts a process that achieves a goal. However, there are also situations that are harder to reconcile at this very high level of abstraction. First, in a service-as-process view a "service" could very well represent a large workflow quite explicitly spanning multiple physical computing entities that achieves a clear goal. On the other hand, a single service in the service-as-endpoint view (such as, a UDDI service itself) clearly encompasses many processes, each of which achieves different goals (e.g., typical client goals such as finding a business entity with certain qualifications, administrative goals such as deleting a business entity record, etc.). To put it succinctly, in the service-as-endpoint view an agent *is* a service, while in the service-as-process view an agent *provides* a service or services.

The syntactic key to linking these disparate models is the use of metadata annotations to indicate certain places where the conceptual models overlap. We support both structured and unstructured metadata. While the former lends itself to better navigation and querying, the latter is still very much in use in the bioinformatics domain. We also believe that service directories can be used by individual users to attach their personal experiences about services (such as accuracy of results, reliability or trust), and therefore, metadata may be used either to support automatic processing in an open manner, or simply for human reading and analysis.

3.1 UDDI and WSDL

Figures 1 and 2 present summaries of the information model for UDDI and WSDL. Both present a view of service-as-endpoint(s), but at different levels of detail. While they are independent information models, they are linked together indirectly through the TModel construct, used to register a WSDL document in a UDDI registry [2], and directly via a metadata assertion.

The core information model in UDDI consists of four concepts, BusinessEntities, BusinessServices, BindingTemplates, and TModels. In UDDI, there are actually two uses of the term service: a Business Service and a technical service. A BusinessService is "descriptive information about a particular *family* of *technical services*" (emphasis ours) [13], a technical service then being represented by a BindingTemplate, representing an entity with a single specific access-point/endpoint. The BusinessService structure is "oriented toward auxiliary information about ... services", simply allowing one "the ability to assemble a set of services under a common rubric" [13].

With respect to integrating UDDI and WSDL, "each bindingTemplate structure represents an individual Web service." [13]. From the WSDL standpoint, a Web Service is a "collection of ports", and thus of "a collection of related endpoints" [16]. WSDL documents contain definitions at several levels of abstraction. "This allows the reuse of abstract definitions: *messages*, which are abstract

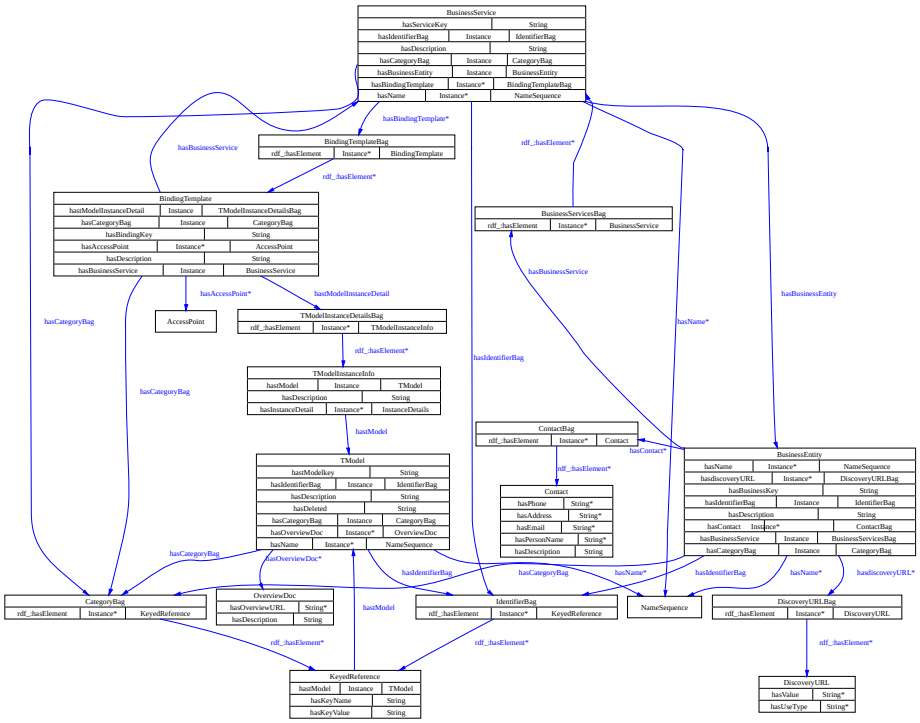

Fig. 1. Partial UDDI information model

descriptions of the data being exchanged, and *port types* which are abstract collections of *operations*. The concrete protocol and data format specifications for a particular port type constitutes a reusable *binding*. A *port* is defined by associating a network address with a reusable binding, and a collection of ports define a *service*" [16] (emphasis ours).

UDDI provides no data structures to represent either the abstract or concrete details, but only a standard way to annotate that a BindingTemplate's endpoint does in fact implement a particular WSDL binding (and thus, a port type, a set of operations, and input/output/fault messages). In the WSDL information space, then, operations and messages are the key to anchor semantic annotations, as exploited by DAML-S and BioMOBY. It is precisely in these abstractions that WSDL is describing *processes* (or at least the atomic processes) that might achieve goals, and thus provides a bridge to service-as-process worldview. This is why the core model of out integrative service directory combines both UDDI and WSDL, otherwise we could not store annotations for goal-directed processes as well as physical endpoints.

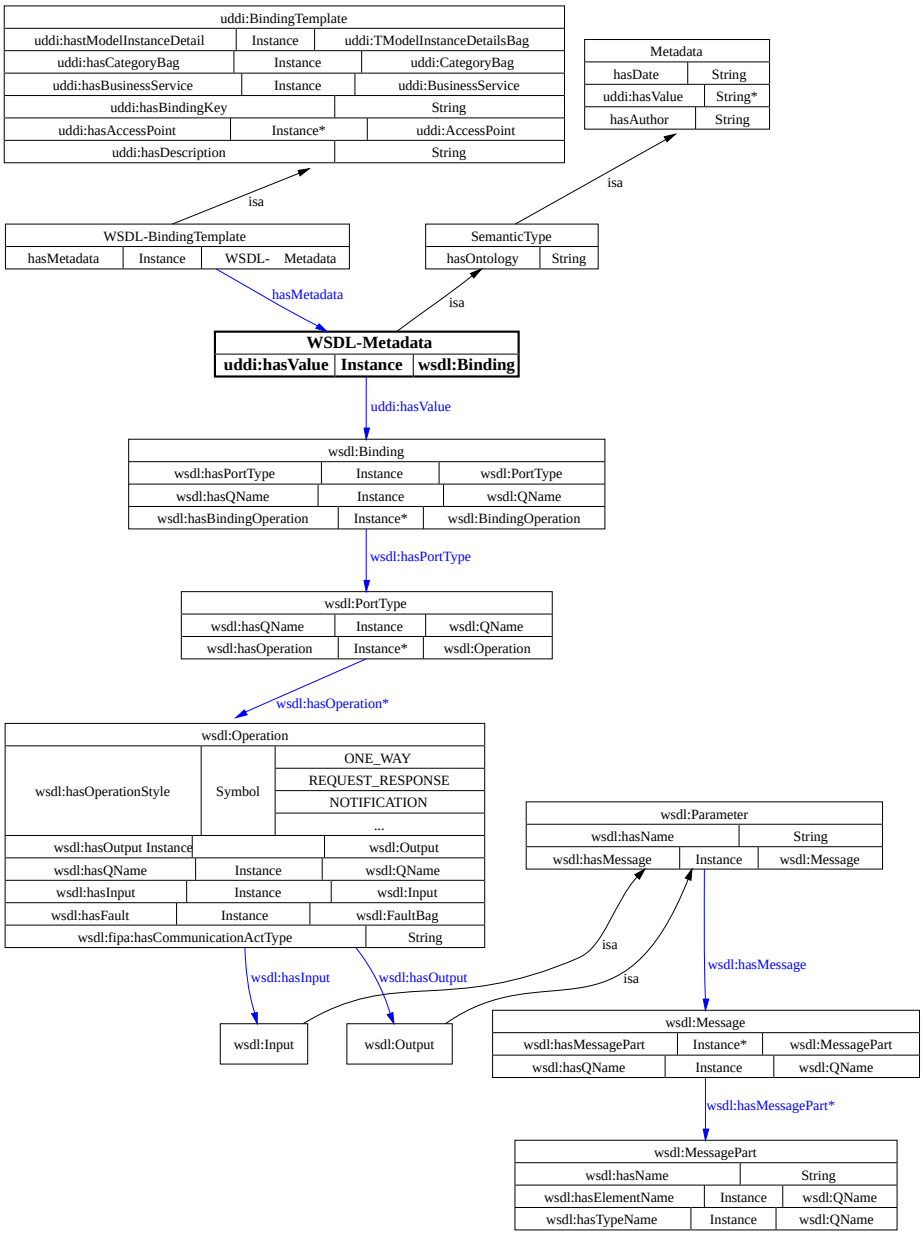

Fig. 2. Partial WSDL information model showing explicit link to UDDI

3.2 DAML-S

DAML-S attempts a full description of a service from the point of view that it is some process that can be enacted to achieve a goal. A full DAML-S service

description incorporates three component perspectives: an abstract description of the service from the AI planning-based "inputs, outputs, preconditions, and effects" view (the service profile); the workflow view of the more primitive services needed to accomplish a complex goal (the service process); the mapping of the atomic parts of this workflow to their concrete WSDL descriptions (the service grounding). The service grounding component is mapped directly to the WSDL document that we represent directly. With respect to the service process component, we currently support only atomic processes that can be mapped to WSDL operations. At its most complex, the DAML-S process view may be nested and include an explicit control model in order to monitor, alter, and possibly terminate the execution of a non-atomic service. One might then go on to draw parallels to Web Service process representations such as WSFL and BPEL4WS [8, 3] and their associated standards, but this is beyond the scope of this paper.

Figure 3 shows our representation of the DAML-S service profile component. A DAML-S service may be classified semantically by a ServiceCategory, and may have input and output ParameterDescriptions that are annotated with semantic DAML class restrictions, while also being directly linked to WSDL MessageParts. We attach an atomic DAML-S service process representation as metadata to a WSDL operation representing that atomic process, and also attach the approriate DAML-S ParameterDescriptions as metadata to the appropriate WSDL MessagePart. When registering a DAML-S service using the DAML-S interface (Section 4), we create a simple UDDI business service, with one Binding Template expressing a simple Web Service whose only port type contains a single operation with input and output messages containing the appropriate message parts.

3.3 Biomoby

BIOMOBY [14] 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 take 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. Essentially, what we have is a simple atomic DAML-S service. The trick is to attach the approriate semantic metadata so that one can recreate BIOMOBY's service registration and find service APIs. Figure 4 shows one way of doing this, using a DAML-S profile.

4 Multiple Service Directory Interfaces to a Triple Store

Based the information model described in Section 3, we have designed and implemented a service directory that supports multiple interfaces, which we now describe. Figure 5 and 6 depict excerpts of class and collaboration diagrams. We see that a service directory implements a series of factory methods to create

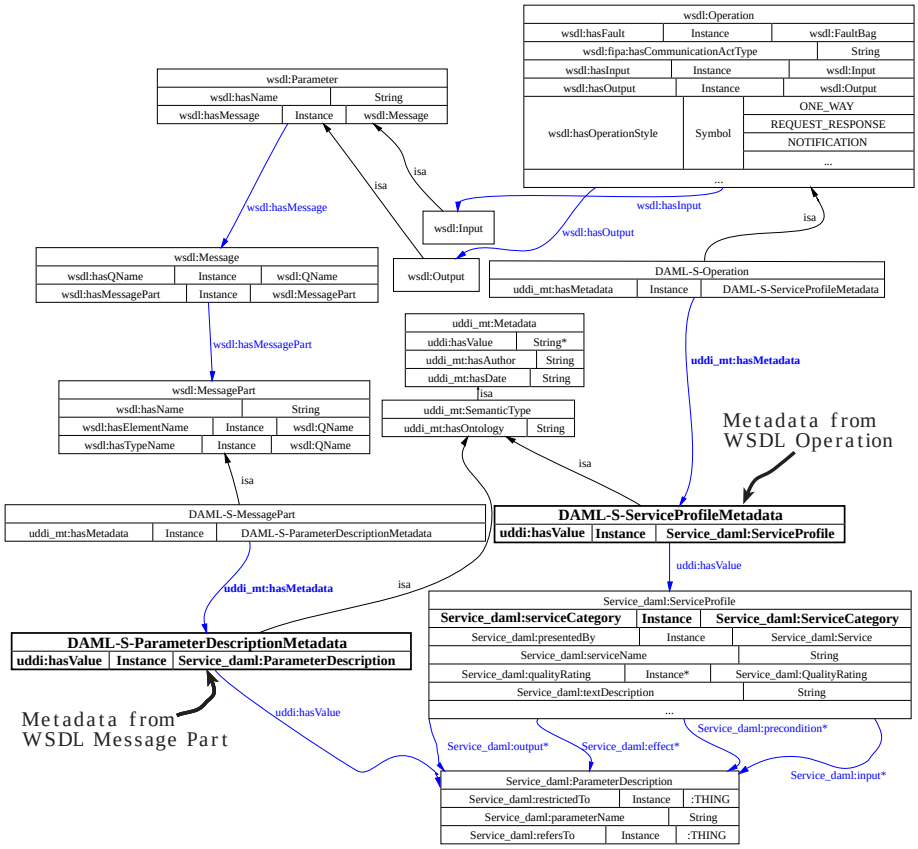

Fig. 3. Semantic descriptions of operations and parameters linked to DAML-S profile

instances of interfaces to the triple store. The triple store is passed as 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 [7]. We currently support the following interfaces to the service directory.

The `InquiryUDDI` and `PublishUDDI` interfaces are full implementations of the inquiry and publish interfaces of UDDI version 2, which relies on the information model defined by the ontology presented in Section 3.1.

The `WSDL` interface allows the registration of a WSDL file in the registry, resulting in all its contents to be explicitly represented according to the ontology of Section 3.1.

The `PublishMetadata` interface allows metadata to be attached to any entity of the information model, whereas the `InquiryMetadata` allows the discovery of such entitities according to metadata. These interfaces were designed in a similar style to the UDDI interface, so that UDDI clients could easily be extended

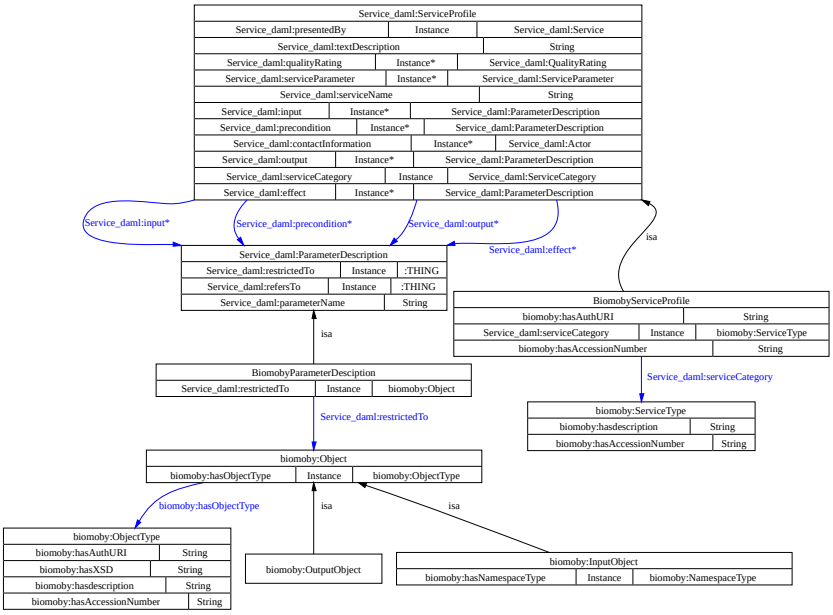

Fig. 4. Attaching BioMOBY semantic information to a DAML-S service profile

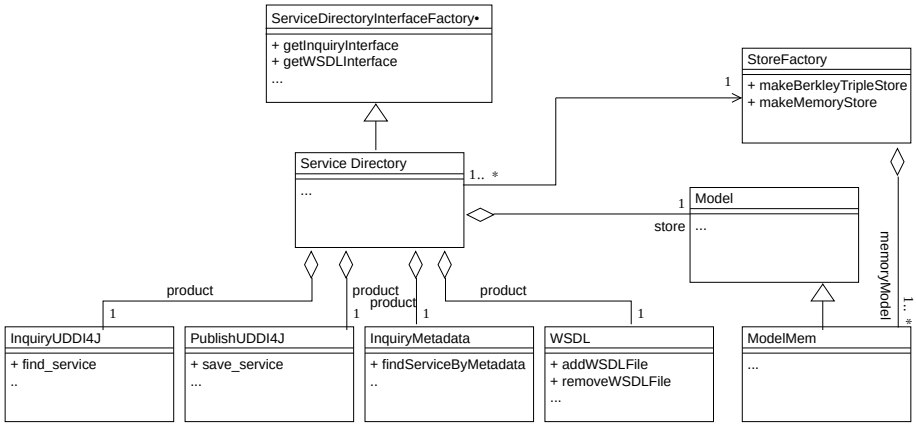

Fig. 5. Class Diagram

to support such features. As an illustration, the following methods allow the attachment of metadata to a UDDI business service, identified by its key, and the retrieving of a service according to metadata.

```
Metadata addMetadataToBusinessService (String serviceKey,
```

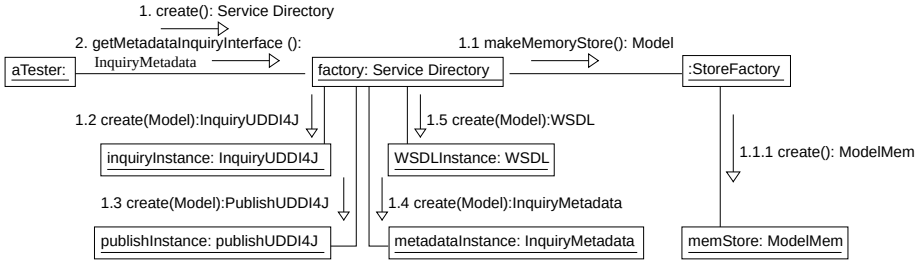

Fig. 6. Collaboration Diagram

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

Metadata is particularly useful to attach semantic descriptions to message parameters (as specified by WSDL files); therefore, we also provide methods for both registring and querying such information for WSDL message parts.

```
Metadata addMetadataToMessagePart (String messageNamespace,
                                   String messageName,
                                   String partName,
                                   Metadata metadata);
MetadataDetail getWSDLMessagePartMetadata (String messageNamespace,
                                           String messageName,
                                           String partName);
```

The `Lease` interface provides support for soft-state based registrations [5], also known as leases in Jini [10], according to which services are registered for a period of time, after which, in the absence of a lease renewal, registrations will no longer be visible through the inquiry interface. It belongs to the management policy (to be discussed in Section 5.2) to decide what action to take in such circumstances.

There is some partial support for the DAML-S ontology in the form of the *service profile*, which describes the functionality the service offers, its semantic inputs and outputs, and the *service grounding*, which provides a mapping to WSDL message descriptions; we currently only focus on atomic processes. We allow a service profile and grounding to be registered and to be discovered using a Java API that we conceived for DAML-S and now explain. For each class of the DAML-S ontology, a Java interface (and its implementation) is defined. In order to register a service, the programmer needs to construct the corresponding Java objects, and submit them using the API. (We are also defining a parser able to construct such Java objects directly from complete DAML-S files.) The querying interface is similar, in the spirit of the UDDI4J library for UDDI: programmers

can construct partial service objects, i.e., with some field not defined, which correspond to a query that it executed over the triple store.

The BioMOBY interface provides some functionality similar to BioMOBY central: it allows a service and its semantic inputs and outputs to be registered, and it allows services to be discovered according to their semantic inputs and outputs.

Finally, we are also providing a more direct interface to the triple store, allowing users to query the service directory using the RDQL query language [7]. We are also finalising the API to allow users to store triples in the triple store: the difficulty here is to ensure that a triples submissions do not corrupt the invariant of the data model we have adopted.

We have defined a generic mechanisms that allows us to attach metadata to resources related to service descriptions. As an illustration of this mechanism, we show how it can be used to register services described according to the myGrid ontology [15]. In this context, services are given a profile specifying which task they perform (`perform_task`), which resources they use (`uses_resources`), what function they consist of (`is_function_of`), and what method they use (`uses_method`). We have defined a convenience function that expects values for each of these relationships and asserts the corresponding triples in the triple store. A relevant excerpt of a result triple store linearisation is displayed in Figure 7, with `A8` an anonymous ode denoting a service and `A3` the node denoting the "myGrid profile". Symmetrically, a convenience function for searching across such attributes is also provided.

```
<rdf:RDF ...>
  <rdf:Description rdf:nodeID='A8'>
    <rdf:type rdf:resource='http://www.mygrid.ecs.soton.ac.uk/uddi.rdf#BusinessService'/>
    <uddi:hasServiceKey>e88b9a57-04bd-481d-888b-dde169b9c48b</uddi:hasServiceKey>
    <uddi:hasName rdf:nodeID='A4'/>
    <uddi:hasDescription>a test service</uddi:hasDescription>
    <uddi:hasBindingTemplate rdf:nodeID='A0'/>
    <uddi:hasMetadata rdf:nodeID='A3'/>
  </rdf:Description>

  <rdf:Description rdf:nodeID='A3'>
    <rdf:type rdf:resource='http://www.mygrid.ecs.soton.ac.uk/mygrid.rdf#Profile'/>
    <rdf:value rdf:resource='http://www.mygrid.ecs.soton.ac.uk/mygrid.rdf#Pe88b9a57-04bd-481d-888b-dde169b9c48b'/>
    <metadata:hasDate>Wed Apr 23 20:39:19 BST 2003</metadata:hasDate>
    <metadata:hasAuthor>Luc Moreau</metadata:hasAuthor>
  </rdf:Description>

  <rdf:Description rdf:about='http://www.mygrid.ecs.soton.ac.uk/mygrid.rdf#Pe88b9a57-04bd-481d-888b-dde169b9c48b'>
    <mygrid:performs_task>retrieving</mygrid:performs_task>
    <mygrid:uses_resources>SWISS-PROT</mygrid:uses_resources>
    <mygrid:is_function_of>BLAST</mygrid:is_function_of>
    <mygrid:uses_method>method</mygrid:uses_method>
  </rdf:Description>
 </rdf:RDF>
```

Fig. 7. Excerpt of an RDF Linearisation for a myGrid profile (**Values must be more convincing!!**)

All these interfaces have been specified and implemented in Java. The documentation of these interfaces, the ontolgies and an example an RDF linearisation of a triple store containing a complete service descriptions are available at: `http://www.mygrid.ecs.soton.ac.uk/service-directory/`

5 Discussion

5.1 Extending Existing Interfaces with Semantic Discovery Support

In Section 4, we have presented all the interfaces that currently provide access to our general information model. Some of them preserve compatibility with the existing standards UDDI, and ensure inter-operability within the Web Services community. Others, such as the direct 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. While such a functionality is very useful, its radically different nature, does not offer a smooth transition for clients implementors wishing to adopt semantic discovery.

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 `find_service` method of the UDDI interface to support queries over metadata that would have been attached to published services. In the method specification below, a new criterion `metadataBag` for identifying services is accepted, containing a set of metadata that a service must satisfy; all other method arguments remain identical to the UDDI specification.

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

5.2 Management policy

In the paper, we have mentioned the use of a policy to define how to manage the service directory in some specific circumstances. While such ideas are still under investigation, our design and implementation are intended to accomodate them. Currently, we foresee the use of policies to support leases, explicit reasoning, and configuration of the system in a distributed system. We now discuss these specific points in turn.

One can imagine several ways of handling the expiry of a lease. A service that has not been renewed could be removed from the directory, or could be made invisible to queries, or even pro-actively contacted to ascertain its existence. Each individual solution is plausible in some scenario, but can difficultly be imposed

as the only way of systematically handling such a situation. Our solution is to rely on a policy that would specify how to react to a lease expiry event.

In our current implementation, we have hard-coded the fact that when a DAML-S atomic process is registered, we construct the necessary UDDI entities, such as a BusinessService with a ServiceKey, so that they can also be retrieved from the UDDI interface; symmetrically, when a UDDI service is registered, we also automatically construct a DAML-S profile for all of its operations. Such decisions need not be hard coded by the implementation of the service; instead, they can be made explicit in the form of inference rules also specified in a management policy.

Finally, we have identified different useful ways of deploying a service directory. First, it can be deployed as a standalone service, available to any client according to its security policy, and presenting the set of interfaces required by its deployer. Second, users could deploy it as a "proxy" to a publicly available service directory for which they do not have write access [9]; the proxy would typically tunnel queries to an existing service directory, but would hold any metadata information about registered services, and would therefore act as a personalised service directory. Finally, the service could also federate entries from multiple service directories. Such configurations, and many others, can only be decided at deployment time according to the specific users' needs. We believe that they must be described in policy files which can be loaded at initialisation time.

6 Conclusion and future work

In this paper, we have discussed an ontology that encompasses the information models of several standards related to service discovery (UDDI,WSDL) and emerging techniques to semantically describe services (DAML-S,BioMOBY). Such an ontology facilitates the encoding of semantics-enriched service descriptions in a triple store, and the triple store is made accessible through multiple interfaces supporting the above standards. The ontology allows us to offer powerful and unparalleled querying capabilities through the RDQL query language. The ontology also allowed us to explore and relate the different notions of services underlying the different approaches. Finally, we have designed extensions to the standard interface UDDI to provide semantic capabilities, hereby offering a smooth transition to semantic discovery for UDDI clients.

In the short term, we want to support further interfaces including the Grid MDS (Monitoring and Discovery Service), and the emerging JAXR offering an information model for both ebXML and UDDI. We also intend to rely on a ontology for bioinformatics services [15] to describe domain-specific properties of services. Currently, all our policies have been hard-coded and we want to look at specific policy languages to support the management of our service directory.

References

1. Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
2. John Colgrave and Karsten Januszewski. Using wsdl in a uddi registry, version 1.08. http://www.oasis-open.org/committees/uddi-spec/doc/bp/uddi-spec-tc-bp-using-wsdl-v108-20021110.htm, 2002.
3. Francisco Curbera, Yaron Goland, Johannes Klein, Frank Leymann, Dieter Roller, Satish Thatte, and Sanjiva Weerawarana. Business process execution language for web services. http://www.ibm.com/developerworks/library/ws-bpel/, 2002.
4. DAML-S Coalition:, A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In *First International Semantic Web Conference (ISWC) Proceedings*, pages 348–363, 2002.
5. 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.
6. Java API for XML Registries (JAXR). http://java.sun.com/xml/jaxr/, 2002.
7. Jena semantic web toolkit. `http://www.hpl.hp.com/semweb/jena.htm`.
8. Frank Leyman. Web Services Flow Language (WSFL). Technical report, IBM, May 2001.
9. Simon Miles, Juri Papay, Vijay Dialani, Michael Luck, Keith Decker, Terry Payne, and Luc Moreau. Personalised grid service discovery. Technical report, University of Southampton, 2003.
10. Scott Oaks and Henry Wong. *Jini In a Nutshell*. O'Reilly, 2000.
11. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Importing the Semantic Web in UDDI. In *Web Services, E-Business and Semantic Web Workshop*, 2002.
12. Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2001.
13. Universal Description, Discovery and Integration of Business of the Web. `www.uddi.org`, 2001.
14. MD Wilkinson and M. Links. Biomoby: an open-source biological web services proposal. *Briefings In Bioinformatics*, 4(3), 2002.
15. 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.
16. Web Services Description Language (WSDL). `http://www.w3.org/TR/wsdl`, 2001.
17. XML Protocol Activity. `http://www.w3.org/2000/xp`, 2000.