Performance Analysis of a Semantics Enabled Service Registry

Weijian Fang Sylvia C. Wong Victor Tan Simon Miles Luc Moreau
School of Electronics and Computer Science
University of Southampton
wf@ecs.soton.ac.uk

Abstract

Service discovery is a critical task in service-oriented architectures such as the Grid and Web Services. In this paper, we study a semantics enabled service registry, GRIMOIRES, from a performance perspective. GRIMOIRES is designed to be the registry for myGrid and the OMII software distribution. We study the scalability of GRIMOIRES against the amount of information that has been published into it. The methodology we use and the data we present are helpful for researchers to understand the performance characteristics of the registry and, more generally, of semantics enabled service discovery. Based on this experimentation, we claim that GRIMOIRES is an efficient semantics-aware service discovery engine.

1 Introduction

Service discovery is a critical task in service-oriented architectures such as the Grid and Web Services. The challenges are the following: (1) since the heterogeneity of information models is inherent to such large scale, distributed systems, it is important to guarantee interoperability among different information models in service discovery; (2) given that the service usage information is a valuable search criterion, it is very helpful for both service providers and service consumers to have the capability to annotate a published service with such usage information as well as to search a service based on a certain annotation; and (3) since there are potentially a large number of services advertised, it is difficult to efficiently and precisely locate the most suitable one based on user provided criteria.

To address these challenges, we are designing and developing a registry called GRIMOIRES (www.grimoires.org) for the myGrid project (www.mygrid.org.uk) and the Open Middleware Infrastructure Institute (www.omii.ac.uk). To address the first challenge, GRIMOIRES supports *semantic discovery* by representing all published information in the form of RDF triples. Thus potentially they can be reasoned over using ontology models to enforce the interoperability among various service description models. To address the second challenge, GRIMOIRES supports metadata attachment to service adverts by both service providers and consumers [9]. Further descriptions of services, such as the quality of a service, or the semantic type of the input of a service operation, can be added as annotations to facilitate future service discovery. We argue that metadata annotation can provide more informative descriptions of published services in an extensible way, so that discovery based on metadata helps improve the efficiency and accuracy of service discovery, thus addressing the third challenge.

In this paper, we study the GRIMOIRES approach from the perspective of performance. In particular, we analyze the scalability of GRIMOIRES against the amount of information published into it.

We have conducted extensive experiments to study the performance of GRIMOIRES. The experiments include (1) the performance comparison between GRIMOIRES and jUDDI (ws.apache.org/juddi), an open source standard UDDI registry, to study the overhead of representing published information in the form of RDF triples and supporting metadata-based discovery; (2) the memory usage of GRIMOIRES; (3) GRIMOIRES' performance using various persistent stores; (4) GRIMOIRES' performance under the security framework specified by WS-Security [8] and related specifications; and (5) the performance breakdown of semantic service publication and metadata-based discovery.

Our contributions are:

1. We report on the performance of a service registry that supports semantic discovery. Since the registry is the core component of a service-oriented architecture, our performance analysis

is of significant interest for researchers in this area to understand the performance characteristics of the registry and semantics enabled service discovery. To the best of our knowledge, this is the first performance report of a semantics enabled service registry.

2. We propose a methodology to study the scalability of service publication and inquiry performance with respect to the registry data size. The same methodology is also applied to measure the performance of attaching, updating, and deleting metadata to published services.

3. Based on such performance analysis, we demonstrate that the GRIMOIRES approach is an efficient semantic service discovery solution in a service-oriented architecture.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents GRIMOIRES' approach towards semantics-enable service publication and discovery. Section 4 introduces our methodology to investigate the registry performance. Section 5 presents the experiment results and gives a detailed analysis. Finally, we give the conclusion and future work.

2 Related work

The UDDI service directory (Universal Description, Discovery, and Integration) [6] has become the defacto standard for service discovery in the Web Services Architecture. Service queries are typically white or yellow pages based: services are located based on a description of their provider or a specific classification (taken from a published taxonomy) of the service type. Service descriptions in UDDI are composed from a limited set of high-level data constructs (Business Entity, Business Service, etc.) that can include other constructs following a rigid schema. However, UDDI does not provide enough support to annotate a published service, or to discover a service based on annotated metadata. While UDDI provides ways to attach a form of metadata to a service advert through the use of tModels, by recording the URL of metadata in tModels, the metadata itself is not saved in the registry. Thus UDDI lacks the capability of inquiry by the content of any metadata. UDDI also does not allow service consumers to attach metadata to a published service. The WSDL language is the standard way to define the technical interface of Web Services. In UDDI, tModels are used to link a WSDL interface to a service by recording the URL of an external WSDL file. UDDI does not provide the capability for users to annotate an element of its technical interface, or to search a service based on a certain characteristic of its technical interface. For instance, UDDI does not allow users to qualify the input of one operation of a Web Service with a semantic type. Such a semantic type would provide a description of the input in terms of the application's semantics rather than the type encoded in the SOAP message.

There are several other service description models than UDDI. For instance, OWL-S [10], formerly DAML-S, uses the OWL web ontology language to describe a Web Service from three component perspectives: what it does (service profile), how it works (service model), and how to access it (service grounding). Biomoby [11] defines a service as an atomic process or operation that takes a set of inputs and produces a set of outputs. In Biomoby, the service, inputs, and outputs can all take semantic types; inputs and outputs also have syntactic types.

The Globus Toolkit (www.globus.org) provides the Monitoring and Discovery System (MDS), consisting of a suite of Web Services, to monitor and discover resources and services on Grids. MDS focuses on the mechanism to disseminate and gather information on Grids rather than the information model to describe services or resources. Each information source publishes information in XML according to some schema. The GLUE schema [12] is used for compute information. But the authors of the information sources or the grid resources, have the freedom to define their schema, so that arbitrary XML data can be published to describe service profiles and states. The XPath language is used for inquiry. We consider MDS as a complement to GRIMOIRES in terms that MDS can be used as an automatic service information collector for GRIMOIRES registry.

3 GRIMOIRES approach

GRIMOIRES is a UDDIv2 compliant registry for Web Services, which itself is implemented as a Web Service, shown in figure 1. The clients interact with GRIMOIRES through sending and receiving SOAP messages. In addition to the UDDIv2 interface, GRIMOIRES also provides some other interfaces, such as metadata and WSDL, which allow clients to publish and inquire metadata and WSDL-related data, respectively. An access control layer within GRIMOIRES enforces fine-grained access control for each published entity, which could be a UDDI service, a piece of metadata, or a WSDL description. All the data published through various interfaces are represented as RDF triples internally, which can be queried and reasoned over in a uniform way. The published data, i.e., the RDF triples, can reside in a

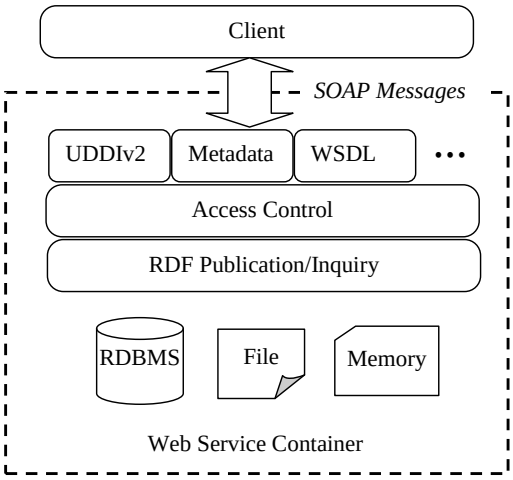

Figure 1: The architecture of GRIMOIRES

database, a file, or simply in memory, to allow deployers to balance persistence with performance.

From the functionality perspective, GRIMOIRES has the following features.

Registration of semantic descriptions GRIMOIRES has the ability to publish and inquire over metadata. Metadata are extra pieces of data giving information about existing entities in the registry. Currently, entities to which metadata can be attached are UDDI BusinessEntity, BusinessService, tModel, BindingTemplate, and WSDL operation and message part.

A piece of metadata is in the form of an RDF triple: the subject is the entity to be annotated, the predicate is the type of the relationship, and the object is the value. The metadata value can be a string, a URI, or structured data in RDF. For example, to describe the quality of a service, the pair (mygrid:NumericRating, 8.5) can be used for the relationship and value of metadata attachment, which assigns a rating of 8.5 to the service.

A unique key is assigned to every piece of metadata published. Therefore, metadata attachments can be updated without republishing the service. This presents an efficient way of capturing ephemeral information about services that changes often, such as the current load of a service.

Multiple metadata attachments There is no limit to the number of attachments each entity can have. Since each piece of metadata has its own unique key, it can be updated without republishing other metadata attached to the same entity.

Third party annotations The ability to publish metadata is available to both service providers and third parties. This provides the flexibility of allowing users with expert knowledge to enrich service descriptions in ways that might not be conceivable to the original publishers. For instance, users can provide their personal ratings on services.

Inquiry with metadata Multiple search patterns are supported in GRIMOIRES. An entity can be found according to a metadata expressed as either a sequence of (type, value) pairs or an RDQL statement. The operation returns a list of entities annotated by metadata matching the query. To support queries over both metadata and non-metadata (such as the name of a service), we have extended the UDDI service finding operation with similar metadata query facility.

Signature based authentication UDDIv2 and v3 specifications rely primarily on the use of authentication tokens to authenticate users for publisher API calls. In implementations such as jUDDI [4], this is generally achieved through a username/password credential scheme. However, this authentication method does not scale well for most Grid environments, which typically use certificate-based authentication schemes. The OMII framework provides an implementation of SOAP message signing and verification in accordance with WS-Security standards [8]. When deployed within the OMII container, GRIMOIRES can extract the Distinguished Name (DN) from the submitted X509 client certificate for authentication purposes. Incorporating signature usage in this way makes it easier to integrate GRIMOIRES into existing Grid security infrastructures, as well as providing an important building block for single sign-on capabilities, an important requirement for many Grid applications.

Access control Access control is on the basis of authenticated identity, and is applied on the granularity of each registered data entry, e.g., a service, a WSDL file, or a piece of metadata. The access control assertions are represented as metadata and are attached to the corresponding data entries.

4 Experiment methodology

In a service registry, two fundamental operations are service publication and service discovery. It is interesting to know the overhead of individual publication and discovery operations with respect to the data size of the registry. We want to investigate to what extent the publication and discovery overheads are affected when an increasing amount of data is registered into GRIMOIRES.

In this test, we use UDDI data model to describe a Web Service. In order to publish a UDDI service, three steps need to be performed:

1. Publish a UDDI business hosting the service by invoking the `save_business` operation,

2. Publish a UDDI tModel recording the URL of the service's technical interface by invoking the `save_tModel` operation, and

3. Publish a UDDI service by invoking the `save_service` operation, referring to the previously published business and tModel.

Two steps are involved in querying a UDDI service:

1. Find a service by its name by invoking the `find_service` operation, and

2. Retrieve the service detail by invoking the `get_serviceDetail` operation, whose input is the service key returned by `find_service`.

All the above operations are defined in the UDDI API specification. Each step can be either a Web Service invocation which incurs a pair of SOAP messages (request and response), or a Java method invocation in the business logic test.

In the test, we repeat the following procedure:

1. Publish 100 services. Each published service has its unique service description.

2. Among all the services currently registered in GRIMOIRES, randomly choose 100 services to query. To do this, the test client maintains a name list for all the registered services.

Using the above method, we measure the service publication and discovery performance of GRIMOIRES against the registry data size under various settings, to investigate the scalability of GRIMOIRES with respect to the number of services published.

5 Performance analysis

Except where stated otherwise, GRIMOIRES runs on a computer of Intel P4 3GHz CPU, with 2GB memory. GRIMOIRES is a Java application. Sun JDK SE 1.4.2 is used. The JVM heap size is set to 1G when running GRIMOIRES either as a standalone application in the business logic test, or as a Web Service deployed in a Web Service container such as Apache Axis. In these experiments, we follow the methodology described in the previous section.

5.1 GRIMOIRES vs. jUDDI

jUDDI (ws.apache.org/juddi) is an open source standard UDDI registry, but it does not offer specific support for metadata-based discovery. By comparing the performance of GRIMOIRES and jUDDI, we are able to study the overhead of representing published information in the form of RDF triples and supporting metadata-based discovery.

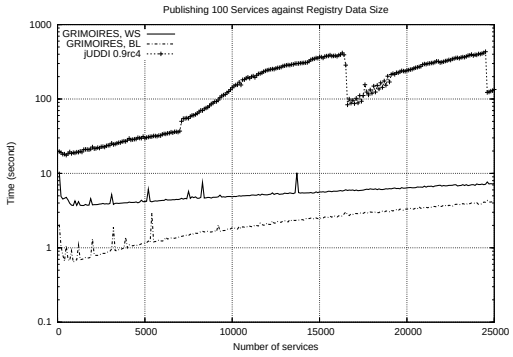

Figure 2: Publication overhead against registry data size

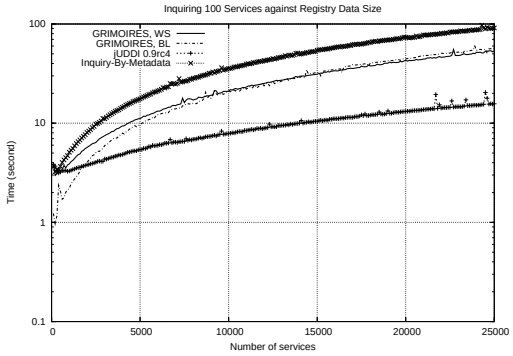

Figure 3: Inquiry overhead against registry data size

Figure 2 and figure 3 show the performance of publication and inquiry of GRIMOIRES and jUDDI against registry data size, respectively. Note that the curves show the overhead of publishing or inquiring 100 services. In the legends, BL denotes the business logic performance that is measured by directly invoking the business logic of GRIMOIRES, whereas WS denotes the Web Service performance that is measured by interacting with GRIMOIRES using SOAP messages. The difference between them is the overhead incurred in the SOAP engine. Ta-

	Publish	Inquire
GRIMOIRES WS	37.8	56.9
GRIMOIRES BL	13.2	40.2
jUDDI	226.3	38

Table 1: The time (in millisecond) to publish and inquire a service when there are 2000 services in the registry

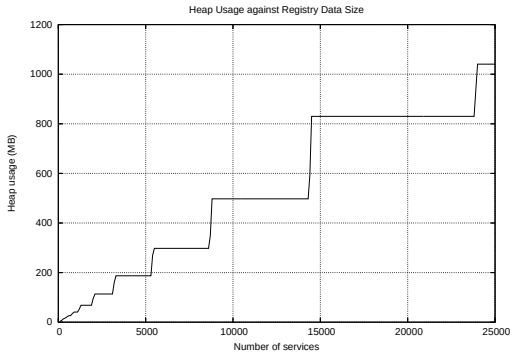

Figure 4: Heap usage against registry data size

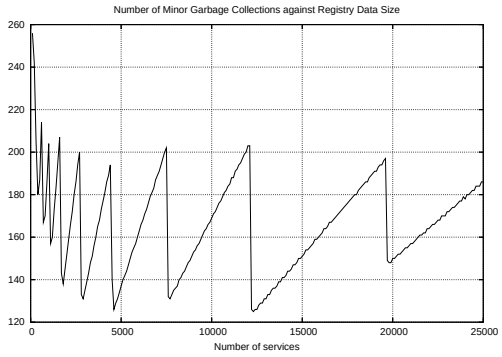

Figure 5: Number of minor garbage collections against registry data size

ble 1 shows the time to publish and inquire a service when there are 2000 services in the registry for various experiment configurations.

GRIMOIRES saves all published information in the memory, which greatly improves the performance compared with saving information to some persistent store such as a database. The memory usage of GRIMOIRES is studied in the later section.

All service information and metadata attached are represented by RDF triples, which are saved to and later retrieved from a triple store within GRIMOIRES. The performance of the triple store is critical to the performance of GRIMOIRES. We are using the latest Jena [5] in GRIMOIRES. We observe to some extent Jena's performance does not scale well when a large number of RDF triples are published.

The publication overhead of GRIMOIRES is much better than that of jUDDI because jUDDI uses a database, PostgreSQL 8.0.1, as a persistent store, while GRIMOIRES stores all information in memory. The performance of GRIMOIRES with persistent stores is discussed in section 5.3.

The inquiry overhead of GRIMOIRES is about twice as that of jUDDI when 5000 services published. To reduce this overhead, we are currently evaluating various RDF stores. Preliminary promising investigations indicate that a more efficient RDF store would bring the inquiry performance of GRIMOIRES to a level comparable to that of jUDDI. Interestingly, the RDF triple store, which introduces some penalty cost, offers GRIMOIRES capabilities that are not supported by jUDDI: in figure 3, we also show the performance of inquiry by metadata. This capability is not available in the UDDI specification.

5.2 Memory usage

Figure 4 shows the heap usage of GRIMOIRES' business logic when using memory as the RDF triple store. It requires about 8 megabytes for 100 services, and 800 megabytes memory for 20000 services. Publishing one service adds 138 RDF statements to the triple store, which consume up about 40 kilobytes memory. We argue that this memory requirement can easily be satisfied on today's servers or even PCs, and the number of services contained in this amount of memory can meet the requirement of most environments. For example, the number of services in myGrid is in the scale of thousands.

Figure 5 shows the number of minor garbage collections incurred when using memory as the RDF triple store. Sun JDK 1.4.2 uses a generational garbage collector by default. A minor collection collects the young generation only, and a major collection also collects the tenured generation. When there is enough allocatable memory in the heap, a typical minor collection takes about one to a few milliseconds, and a typical major collection takes about tens of milliseconds to one second. During the experiment, there are only 19 major collections, 9 of which occurred during the very beginning part of the experiment. Comparing the overhead of a minor collection and that of service publication and inquiry, it can be observed that the disruption caused by garbage collection is not serious under the condition that there is enough allocatable memory in the heap.

5.3 Performance with persistent stores

It is important for the registry to be persistent, so that the contents of the registry is not lost at the moment of a crash. The persistence of GRIMOIRES relies on that of the RDF triple store used in GRIMOIRES. Currently, GRIMOIRES uses Jena as the triple store. Jena can be made persistent in different ways, e.g., by using a relational database, a file-based hash table, or simply a plain text file to store the triples. In this experiment, GRIMOIRES' performance with

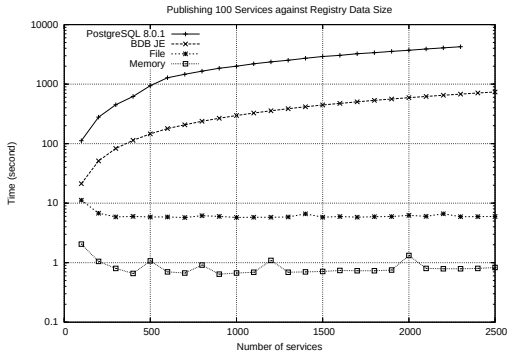

Figure 6: Publication performance with persistent stores

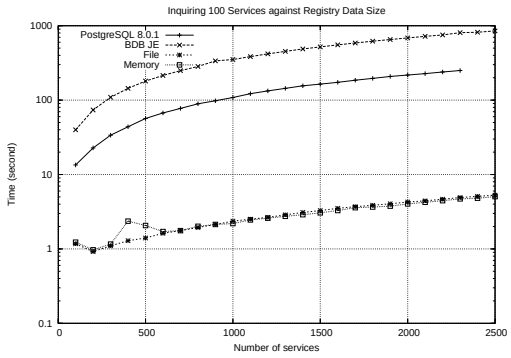

Figure 7: Inquiry performance with persistent stores

the persistence support is measured. Figure 6 and 7 show the publication and inquiry performance of GRIMOIRES with the persistence support, respectively.

In this experiment, we use two persistent stores, PostgreSQL and Berkeley DB Java Edition (BDB JE). PostgreSQL is an open source object-relational database management system widely used in production environments. BDB JE is a data store embedded in applications rather than a relational database engine. PostgreSQL is more efficient than BDB JE for inquiry, but slower than BDB JE for publication. Compared with the performance of GRIMOIRES using memory as the triple store, the publication and inquiry overheads using both PostgreSQL and BDB JE are far more expensive.

Another way for GRIMOIRES to provide data persistence is to use a file as the back end of the triple store. We design a checkpointing scheme in GRIMOIRES, which logs in the file the deleted and added triples before they are actually committed into the triple store. In case of a registry crash, the registry contents before the crash can be restored by replaying the addition and deletion operations based on the logged data in the same sequence as they are logged. This file-based checkpointing scheme does not affect the inquiry performance, but significantly increases the publication overhead.

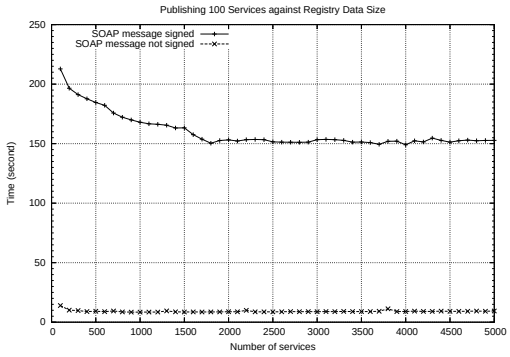

Figure 8: Publication overhead with WS-Security

Table 2 shows the time to publish and inquire a service when there are 2000 services in the registry for various triple store configurations.

5.4 Performance with WS-Security

In this experiment, we measure GRIMOIRES' performance under the framework of WS-Security [8]. The experiment is conducted on a computer using an Intel P4 2.4GHz CPU and 1GB memory.

The OMII environment provides the functionalities of creating X.509v3 certificate based signatures on outgoing client-side SOAP messages, and verifying these signatures on server-side Web Services deployed in the OMII container. Signatures are created and verified respectively through the client-side and server-side handler, which intercept SOAP messages and process them in a way transparent to the client and the service.

We deploy GRIMOIRES in OMII 1.2.0 container and measure its performance adopting the same methodology, with signing of SOAP messages both disabled and enabled. Figure 8 shows the publication overhead, and figure 9 shows the inquiry overhead. As seen from the figures, the signing and verifying of SOAP messages introduces a big overhead.

	Publish	Inquire
PostgreSQL	37,066.3	2,172.8
BDB JE	5,919.9	6,862.4
Memory	13.2	40.2
File	62.4	42.7

Table 2: The time (in millisecond) to publish and inquire a service when there are 2000 services in the registry

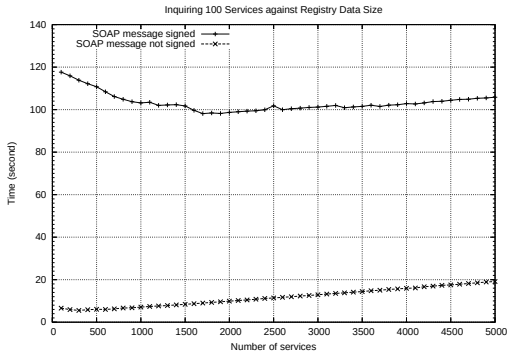

Figure 9: Inquiry overhead with WS-Security

	Publish	Inquire
Signed	1,531.8	987
Not signed	88.3	98.9

Table 3: The time (in millisecond) to publish and inquire a service when there are 2000 services in the registry

Table 3 shows the time to publish and inquire a service when there are 2000 services in the registry. As seen in the table, the signing of SOAP message has a larger impact on the publication overhead than on the inquiry overhead. This is because there are 3 Web Service interactions incurred during the publication, and only 2 during the inquiry. The overhead due to SOAP message signing during the publication, i.e., the difference between when signing is enabled and when signing is disabled, is roughly 1.5 times of that during the inquiry.

5.5 Performance breakdown of semantic service publication and discovery

In this experiment, one piece of metadata is added to each service. All the metadata are of the same annotation type, but with different values. Services are discovered according to the attached metadata.

Figure 10 demonstrates the performance of semantic service publication. We further breakdown the overhead of semantic service publication into several steps, which include publishing the WSDL file that defines its technical specification[1], publishing its general information such as name and organization that are represented by UDDI data model, and attaching metadata to the service. In the experiment, we intentionally publish, republish, and then delete the metadata to measure the metadata-related performance against the registry data size. We observe that

[1] In previous experiments, we do not publish the WSDL file of the service.

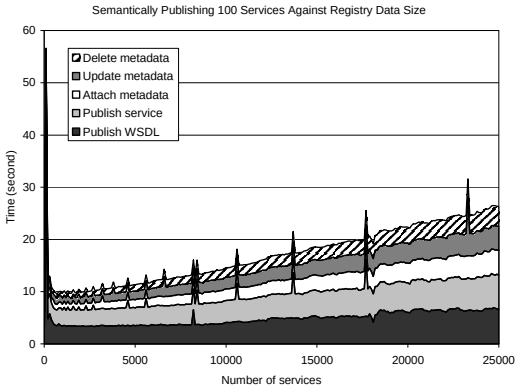

Figure 10: Breakdown of the performance of semantic service publication

Publish WSDL	33.5
Publish service	31.2
Attach metadata	11.7
Update metadata	12.3
Delete metadata	9.1
Inquiry by metadata	80.5

Table 4: The overheads (in millisecond) of metadata-related operations when there are 2000 services in the registry

publishing, updating, and deleting metadata are efficient operations.

In figure 3, we observe that the performance of inquiry by metadata is a little more expensive than the standard UDDI inquiry because the structure of its query statement is more complicated.

Table 4 shows the overheads of metadata-related operations when there are 2000 services in the registry.

6 Conclusion and future work

In this paper, we study GRIMOIRES, a semantics enabled service registry, from the performance perspective. We study the scalability of GRIMOIRES against the amount of information published. Based on this experimentation, we claim that GRIMOIRES is an efficient semantics-aware service discovery engine.

We are investigating the life cycle management of published data in GRIMOIRES. We plan to adopt a soft state protocol, so that the data will expire after its predefined life time. In this way, the stale data in the registry can be removed automatically.

We are also investigating the notification mechanism in GRIMOIRES. When a published entity is updated, GRIMOIRES users who are interested in it,

will receive a notification of change. The notification can be a SOAP message or simply an Email. Users need to register correspondingly in order to receive the notifications. The notification mechanism can be a complement to service discovery, since it can be used to track a service after the service is discovered.

7 Acknowledgements

This research is funded in part by the GRIMOIRES (EPSRC Grant GR/S90843/01) and myGrid (EPSRC Grant GR/R67743/01) projects.

References

[1] GRIMOIRES project. http://www.grimoires.org/.

[2] myGrid project. http://www.mygrid.org.uk.

[3] Open middleware infrastructure institute. http://www.omii.ac.uk/.

[4] jUDDI 0.9rc4. http://ws.apache.org/juddi/.

[5] Jena Semantic Web Framework. http://jena.sourceforge.net/.

[6] UDDI Version 2.04 API Specification. http://uddi.org/pubs/ProgrammersAPI\-V2.04\-Published\-20020719.htm.

[7] WWW Consortium. RDF Primer. http://www.w3.org/TR/rdf-primer/.

[8] Oasis Web Services Security (WSS). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[9] Simon Miles, Juri Papay, Terry Payne, Keith Decker, and Luc Moreau. Towards a Protocol for the Attachment of Semantic Descriptions to Grid Services. *In The Second European across Grids Conference*, Nicosia, Cyprus, pages 10, January 2004.

[10] OWL-S. http://www.daml.org/services/owl-s/.

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

[12] The GLUE schema. http://www.cnaf.infn.it/~sergio/datatag/glue/