Service Oriented Computing and Applications manuscript No.
(will be inserted by the editor)

Performance Evaluation of Semantic Registries: OWLJessKB and instanceStore

Simone A. Ludwig[1], **Omer F. Rana**[2]

[1] Department of Computer Science
University of Saskatchewan, Canada, ludwig@cs.usask.ca
[2] School of Computer Science
Cardiff University, U.K., o.f.rana@cs.cf.ac.uk

The date of receipt and acceptance will be inserted by the editor

Abstract Service discovery is a critical task in distributed computing architectures for finding a particular service instance. Semantic annotations of services help to enrich the service discovery process. Semantic registries are an important component for the discovery of services and they allow for semantic interoperability through ontology-based query formulation and dynamic mapping of terminologies between system domains. This paper evaluates two semantic registries – OWLJessKB implementation and instanceStore – to determine the suitability of these with regards to the query response time and the overall scalability for use in mathematical services. Used for the measurements are the mathematical MONET ontologies.

Key words Service discovery, semantic registries, Web service

1 Introduction

As the Internet provides connectivity and information richness over great distances at any time, it has created a dynamic, open and convenient environment for social and business development. It not only provides the opportunity for new commercial endeavours utilising the Web, but also opens up new opportunities for the old, static, locally based businesses to adopt a new business paradigm and new organisational forms. The Internet has also opened up modes of interaction and dynamic organisational configurations that were previously inconceivable within a wide array of human and business activities. Yet, most of the interactions taking place are managed by humans. The computer science community envisions the future of the interaction on the Internet to be automated and performed by software agents that replace the role of humans.

Service-oriented environments have special characteristics that distinguish them from other computing environments as follows. The environment is dynamic meaning that service providers might become unavailable and new service providers offering new services might go online. This means the environment will change over time as the system operates. The same principle is applied for service consumers. Service consumers can use a service and goes offline immediately. The number of service providers is unbounded, given that service providers can join the environment at any time. Services are owned by various stakeholders with different aims and objectives. There may be incompetent, unreliable or even malicious service providers which make the environment insecure. There is no central authority that can control all the service providers and consumers. In addition, it is assumed that service providers and consumers are self-interested.

The ability to locate services of interest in an open, dynamic, and distributed environment has become an essential requirement in many distributed systems. Traditional approaches to service discovery have generally relied on the existence of pre-defined registry services, which contain descriptions that follow some shared ontology. Often the description of a service is also very limited in existing registry services, with little or no support for problem-specific annotations that describe properties of a service. Semantic registries attempt to overcome this limitation, and provide: (1) a rich semantic description based on an ontology; (2) reasoning capability that can be applied to the semantic description. Semantic matching generally focuses on the problem of identifying services on the basis of the capabilities that they provide. Such matching is provided through a language to express the capabilities of services, and the specification of a matching algorithm between service advertisements and service requests. The use of standards in representation techniques and application-specific concepts play an important part in such service descriptions.

Many applications in computational science make use of numerical algorithms, either developed as part of the project or obtained from third parties – examples include libraries from the Numerical Algorithms Group (NAG). The complexity of such algorithms can vary from simple matrix solving to more complex data analysis – such as clustering or classification techniques. Furthermore, the ability to access such algorithms as Web Services allows easy integration of such capability within an existing application. This also provides a loose coupling between the application and the numerical algorithm. To enable mathematical objects to be exchanged between computer programs in an unambiguous manner (stored in a database or published on the worldwide web), the OpenMath [1] standard was introduced. The main part of OpenMath is the idea of a "Content Dictionary" (CD) – defining a collection of symbols that may be used in a mathematical formula, and their associated meaning and relationships. Essentially, a CD is a small, specialized ontology which may be extended as new symbols are introduced into mathematics. OpenMath may be used to encode the mathematical part of a problem to be solved – such as a differential equation or an integral. Furthermore, OpenMath may be used to describe properties of a mathematical service, by defining sets of symbols which are understood by the service, and those used to represent the result.

The problem of matchmaking for mathematical services was addressed in [2], where the semantics play a critical role in determining the applicability or otherwise of a service and for which we use OpenMath descriptions of pre- and post-conditions. We describe a matchmaking architecture supporting the use of match plug-ins and describe five kinds of plug-in that we have developed to date: (i) a basic structural match, (ii) a syntax and ontology match, (iii) a value substitution match, (iv) an algebraic equivalence match and (v) a decomposition match. The matchmaker uses the individual match scores from the plug-ins to compute a ranking by applicability of the services.

The aim of this paper is to evaluate two semantic registries in the area of mathematical services – instanceStore and OWLJessKB – to compare their scalability and response time. These two registries were chosen for this evaluation as they both provide description logic (DL) reasoning which is an important necessity in the area of mathematical service matching.

Furthermore, both semantic registries were developed explicitly for service matching in the area of mathematical services.

The paper is organised as follows. In Section 2 the measurement setup is described by introducing the target systems and the methodology used. Section 3 shows the measurement results outlining the query and scalability performance. A conclusion of the findings is given in Section 4.

2 Measurement Setup

A set of measurements are described that deals with: (1) the associated query response times, (2) the overall scalability of two semantic registries. Scalability analysis involves increasing the number of services hosted in the registry to 100,000.

2.1 Target Systems

2.1.1 OWLJessKB Implementation For the GENSS project a mathematical matchmaker [1] was developed [3]. It is based on OWLJessKB – a memory-based reasoning tool that may be used over ontologies specified in OWL. To store service descriptions, a MySQL database was used residing on a different machine. OWLJessKB uses the Java Expert System Shell (JESS) [4] as its underlying reasoner. The OWLJessKB implementation loads multiple ontologies (the location of each is specified as a URL) into memory from a remote Web server. Reasoning in this instance involves executing one or more JESS rules over the ontology. JESS makes use of the Rete algorithm [5], which is intended to improve the speed of forward-chained rule systems (this is achieved by limiting the effort required to recompute the conflict set after a rule is fired). However, it has high memory requirements due to the loading of the ontology into the Rete object. Once it is created and set, it is fast to call rules and queries in order to infer the semantic relations of the ontology loaded. A key limiting factor in OWLJessKB is the time to load the ontology into primary memory (RAM), and the total RAM size on the host platform.

2.1.2 instanceStore A DL knowledge base (KB) is made up of two parts: a *terminological* part (the terminology or Tbox), and an *assertional* part (the Abox). Each part consists of a set of axioms. The Tbox asserts facts about concepts (sets of objects) and roles (binary relations between objects), usually in the form of inclusion axioms. The Abox asserts facts about individuals (single objects), usually in the form of instantiation axioms [6]. The instanceStore [7] is a Java application for performing efficient and scalable DL reasoning over individuals. The instanceStore system tackles the problem of requiring vast volumes of individuals by applying a well-known idea in knowledge representation, namely supporting reasoning by means of databases. Assertions over individuals are stored in a database, together with information inferred using a DL reasoner over the position in the ontological taxonomy of their corresponding descriptions. This allows the system to reduce the amount of reasoning to pure terminological reasoning, while maintaining soundness and completeness (provided that no relation between individuals exists). The instanceStore is an implementation having the ontologies and the database (Hypersonic [8]) stored locally (i.e. hosted on one machine).

[1] *http://agentcities.cs.bath.ac.uk:8080/genss_axis/GENSSMatchmaker/index.htm*

2.2 Methodology

Two semantic registry implementations instanceStore version 1.4.1 [2] and OWLJessKB version owljesskb20040223.jar [3] were tested. The test environment included an Intel Pentium III processor 996MHz, 512MB RAM, and a Windows XP Professional, running Java SDK 1.5.0 and Jess 6.1p8.

2.2.1 Measurements - Query response times: Four different query types were chosen. These were the following:
(1) Simple assertion: Find all instances of a class `x`;
(2) Simple assertion: Verify whether instance `x` exists;
(3) Assertion individual: Confirm if constraint `y` is satisfied via a single object;
(4) Assertion aggregate: Confirm if constraint `y` is satisfied for an entire group.

2.2.2 Measurements - Scalability: These measurements involved analyzing the performance of query response times for populated registries, starting with 100 services stored and going up to 100,000 services. Hence, the scalability is evaluated for the change in query response behaviour as additional services were added to the registry. The ontology used for the measurements (MONET ontology) consists of 2031 classes, 78 slots and 10 facets. Different ontology sizes were used having the following number of classes:

– Ontology size 1: 2031 classes;
– Ontology size 1.5: 3046 classes;
– Ontology size 2: 4062 classes;
– Ontology size 2.5: 5077 classes;
– Ontology size 3: 6092 classes;
– Ontology size 3.5: 7107 classes;
– Ontology size 4: 8122 classes.

3 Measurement Results

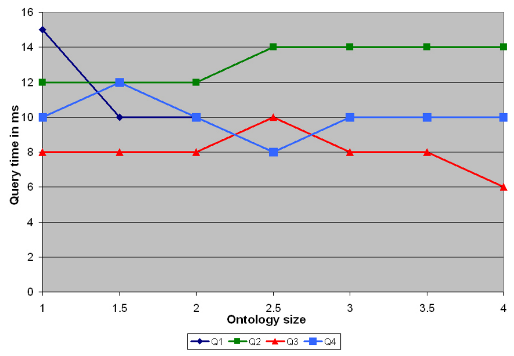
3.1 Query Performance

A set of measurements were carried out having four queries:

– Query 1: Query for services with GAMS classification GamsB (simple assertion finding all instances of a class x);
– Query 2: Query for services with RootFinding algorithm (simple assertion asking whether instance x exists; example query see below);
– Query 3: Query for services running on a single processor (assertion individual);
– Query 4: Query for services whose platform is not parallel (assertion aggregate).

Tested were the response times of varying ontology sizes for both target systems. The maximum heap size in both target system was set to 1 GB. For each query and each target system ten test runs were conducted and the average values were taken.
In Figure 1 the response time of the four queries are shown for the OWLJessKB implementation and the instanceStore. It shows that the response times of the OWLJessKB are much

[2] *http://sourceforge.net/project/showfiles.php?group_id=95954&package_id=102438*
[3] *http://edge.cs.drexel.edu/assemblies/software/owljesskb/*



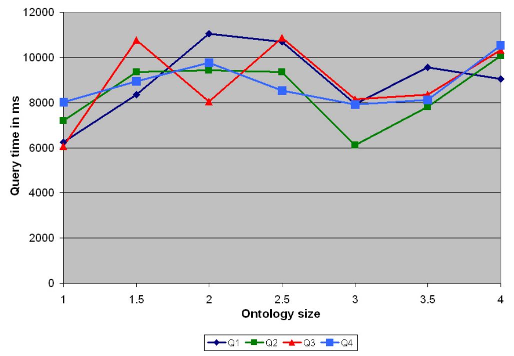


Fig. 1 OWLJessKB Query Performance of OWLJessKB (left; Q1, Q4 times identical after ontology of size 2) and instanceStore (right)

smaller than the ones of the instanceStore. In both figures no increase in query response time for larger ontology sizes can be seen, which means that the ontology size does not increase the query response time. The average deviation (specifying measurement accuracy) is 1.8 ms for the OWLJessKB implementation, and 703 ms for the instanceStore. Comparing the instanceStore with the OWLJessKB implementation regarding the query performance shows that the OWLJessKB implementation performs on average 910 times faster than the instanceStore.

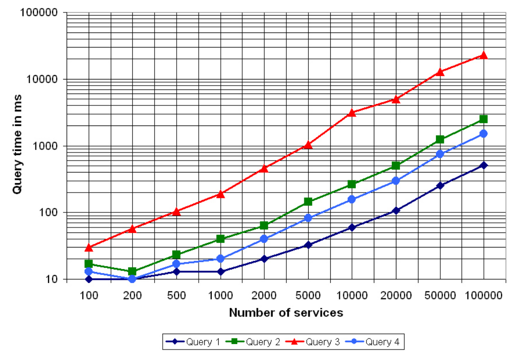
3.2 Scalability Performance

Measurements were taken populating the OWLJessKB implementation and the instanceStore with an increasing number of services/individuals. Ontology size 1 was taken for this set of measurements, with a maximum heap size of 1 GB. Both semantic service registries were populated with 100, 200, 500, 1,000, 2,000, 5,000, 10,000, 20,000, 50,000 and 100,000 services (therefore, Figure 2 use a logarithmic scale). In Figure 2, left hand side, a linear distribution between query time and number of services can be seen. However, the time measurements for 100, 200 and 500 services in Query 1, 2 and 4 are scattered around 10 ms and 20 ms, this is due to the measurement accuracy. It can be seen that Query 3 has the highest query times for the scalability measurements because the query performs an assertion of individuals which is more time consuming than the other queries. In Figure 2, right hand side, a linear distribution can be seen between the time it takes to search for a service, subject to the number of services populated in the registry.

Comparing the OWLJessKB implementation with the instanceStore, regarding the scalability of services stored in the registry, it can be seen that the OWLJessKB implementation performs 1780 times faster than the instanceStore (average for OWLJessKB implementation was 1,374 ms versus 2,445,637 ms for instanceStore).

4 Conclusion

The conducted measurements for the two semantic registries revealed that the scalability of the OWLJessKB implementation is much higher than the instanceStore when searching for



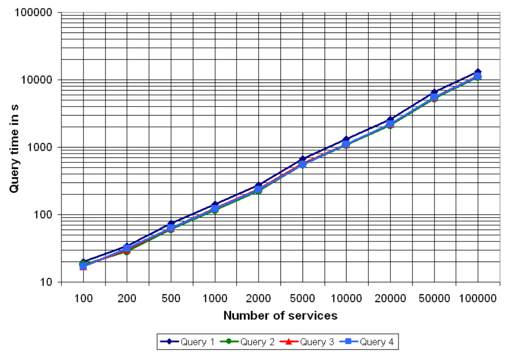


Fig. 2 Scalability of populated services of OWLJessKB (left) and instanceStore (right)

a service. However, the initialization stage, where the ontologies are loaded into memory is much slower. The initialisation of the OWLJessKB implementation uses the Rete algorithm, a drawback of which is its high memory requirements due to the loading of the ontology into the Rete object. Once it is created and set, it is fast to call rules and queries in order to infer the semantic relations specified in the ontology. One other influential factor, which is however a minor one, is that the instanceStore is a local implementation, which means that the ontology, database etc., is stored locally, whereas the OWLJessKB implementation has its ontologies and database stored on one or more remote machines giving it the advantage of a distribution system. In general, the instanceStore performs better as a semantic registry when different ontologies constantly need to be loaded, and the OWLJessKB implementation performs better when a static set of ontologies is used and a large number of services are stored in the registry. It would be interesting for further work to find the maximum amount of services which can be stored in both semantic registries.

References

1. OpenMath Society. Details at: `http://www.openmath.org`.
2. S. A. Ludwig, O. F. Rana, J. Padget and W. Naylor, Matchmaking Framework for Mathematical Web Services, Journal of Grid Computing, vol. 4, no. 1, pp. 33-48, 2006.
3. S.A. Ludwig, O.F. Rana, W. Naylor and J. Padget, "Matchmaking of Mathematical Web Services", In Proceedings of 6th International Conference on Parallel Processing and Applied Mathematics, Poznan, Poland, September 2005.
4. Ernest J. Friedman-Hill, "Java Expert Systems Shell". Available from: `http://herzberg.ca.sandia.gov/jess/docs/61/index.html`.
5. C. Forgy, "Rete: A fast algorithm for the many pattern/many object pattern match problem". Journal of Artificial Intelligence, 19:17–37, 1982.
6. I. Horrocks, U. Sattler and S. Tobies, "Reasoning with individuals for the description logic SHIQ". Proceedings of the 17th International Conference on Automated Deduction (CADE-17), Springer-Verlag, 2000.
7. D. Turi, "Instance Store". Available at: `http://instancestore.man.ac.uk`.
8. The Hypersonic Database (HSQL DB). Available from `http://hsqldb.org/`.