Matchmaking Support for Dynamic Workflow Composition*

Neil Chapman[1], Simone A. Ludwig[1], William Naylor[2], Julian Padget[2] and Omer F. Rana[1]
[1]School of Computer Science/Welsh eScience Centre, Cardiff University
[2]Department of Computer Science, University of Bath

Abstract

Service description and discovery offer complementary challenges, but in both cases, the problem is finding the right trade-off between accuracy and generality that will result in a positive service identification. Discovery systems have historically tended to focus on domain-specific techniques using single sources of knowledge to help classify queries against services, making both maintenance and extension difficult. The primary contribution of this paper is the presentation of a generic brokerage framework based on the use of plug-in components, that are themselves web services. The framework has been developed in the context of the KNOOGLE project, where the focus has been on demonstrating support for (i) the discovery of Grid services for the GridSAM job submission system and (ii) integration with the Taverna workflow enactment system. However, the broker itself is domain independent and it is the multiple user-specified matchmaker plug-ins that act as sources of domain-specific knowledge. The broker collects the results of the matchmakers' comparison of the query and service and then applies a user-specified selection policy to determine the final choice of service. Thus a range of comprehensive packaging of brokerage functionality becomes possible through the use of supplied and user-defined matchers and supplied or user-defined selection policies.

1 Introduction

Matchmaking and brokerage have been a topic of research for some 10–15 years now although it is probably the case that only in the latter half of this period has the issue of the description of the *semantics* of the service come to the fore through the uptake of XML (as a document structuring mechanism) and OWL (as an ontology language). It is notable that in all this time, few if any of the brokers have seen use outside the domain or project in which they were developed. What this seems to indicate is that brokers are typically domain-specific at least, and quite possibly project-specific as well, making them too fragile for redeployment in new problem domains. Furthermore, most of the literature (see section 6) focusses on how just one matching technique may be used to identify candidate services and rank them as meet the query criteria. These observations led us to the conclusion that it should be possible to separate out brokerage function from brokerage operation and that it would be valuable to enable clients to combine the results of *several* matching schemes and then apply a policy stated in terms of rules to determine a preferred service.

Thus, we are in a position to describe a brokerage architecture that makes use of bespoke matchmakers and a high-level selection policy, and that can be used stand-alone or in the context of workflow enactment engines. Given a service description, the broker is able to interact with a registry to find suitable services, apply a range of matchers to compute the degree of match between query and service and subsequently process the service match metrics using a selection policy. The flexibility of the brokerage framework stems from the fact that its architecture involves a component-based approach, which allows the integration of capabilities through the use of Web Services. Thus constructing a new broker becomes a matter of composing a workflow involving: (i) a range of sources of service descriptions; (ii) a range of matching services that will accept a service request and a service description and output a measure of the relationship between the two; (iii) a selection policy that uses the service match result information to determine the best fit; (iv) a service to invoke the selected service and deliver the results. We demonstrate the broker functionality through integration with the GridSAM job submission system. The selection policy is used to determine which of the services meet most closely some specified criteria. However, to include a human in the process, it is also possible for such a decision to be taken by the user. Identifying techniques for describing service properties and queries now becomes significant. Although keywords might help narrow down the search, they do not offer the capability to describe

A preliminary version of this paper appeared in the proceedings of the UK eScience All Hands Meeting, Nottingham, September 2006 – under the title "Service-Oriented Matchmaking and Brokerage".

requirements such as the inputs/outputs wanted. Furthermore a service interface says little about its actual function; for that we require a statement of the *relationships* between the inputs and outputs, or more generally, statements of pre- and post-conditions. We can therefore observe that each service will have a functional interface (describing the input/outputs needed to interact with it and their types) and a non-functional interface (which identifies annotations related to the service made by other users and performance data associated with the service). Being able to support selection on both of these two interfaces provides a useful basis to distinguish between services.

Even when a service (or a composition of a set of services) has been selected, it is quite likely that their interfaces are not entirely compatible. Hence, one service may have more parameters than another, making it difficult to undertake an exact comparison based just on their interfaces. Similarly, data types used within the interface of one service may not fully match those of another. In such instances, it would be necessary to identify mapping between data types to determine a "degree" of match between the services. Although the selection or even the on-the-fly construction of shim services is something that could be addressed from the matchmaking perspective [5], we do not discuss this issue further in this paper.

The essence of the contribution in this paper lies in exposing and discussing the design choices that have been made and considering whether there are any lessons more generally for the architecture of (grid) middleware. The novelty of the architecture described here is that by factoring out a range of control and orchestration issues and placing them in a single customizable component, attention can be focussed on specific and generic matching techniques, with the assurance that the result can be plugged into a brokerage framework, for both standalone use and within workflow enactment systems. The development of the broker software is essentially completed and is now being integrated into the OMII [23] software distribution. We briefly reprise the requirements for brokerage that have driven this work in section 2, then describe in detail the architecture and design of the KNOOGLE broker [22] in section 3. We follow this (section 5 with a description of the main functionality of the broker and an illustration of its integration into the Taverna workflow engine. The paper concludes with some performance results (section 7) and a summary of related work (section 6).

2 Broker Requirements

In two earlier projects, MONET [2] and GENSS [8], the focus was on the discovery and selection of mathematical web services based on semantic matching of the functional relationships between inputs and outputs. One lesson from this work was the potential importance of combining complementary matching technologies in order to reach a better-informed decision about the applicability of a given service. This lead to the implementation (in GENSS) of a matchmaker that utilized several matchers and choose a particular service by combining the match scores from each of the matchers [8]. In KNOOGLE, we have refined and generalized this approach by characterizing the matchmaking and brokerage process in terms of three essential actions:

1. Where to find descriptions of entities to match against — repositories
2. How to match the query against a description — match services
3. How to choose between the matched descriptions — selection policy

To support matchmaking, sufficient input information about the task is needed to satisfy the desired capability, while the outputs of the matched service should contain at least as much information as the task is seeking. Additionally, the task pre-conditions should at least satisfy the capability pre-conditions, while the post-conditions of the capability should at least satisfy the post-conditions of the task. These constraints reflect work in component-based software engineering [15]. Furthermore, given the nature of the problem, it is only very rarely that a task description will match exactly a capability description and so a range of reasoning mechanisms must be applied to identify candidate matches, essentially constructing a *proximity profile*. This results in:

Requirement 1: A plug-in architecture supporting the deployment and utilization of an arbitrary number of matchers.

The second requirement is a consequence of the first: there will potentially be several candidate matches and choosing between them on the basis of *several* matching schemes — rather than just one, where a simple ranking would suffice — some of which may generate conflicting information. This leads to:

Requirement 2: A selection mechanism is required that takes into account all of pure technical, quantitative and qualitative aspects — and user preferences in respect of a service

The user preferences might cover inclusion or exclusion of services with particular properties. See, for example the blacklisting policy in section 4.

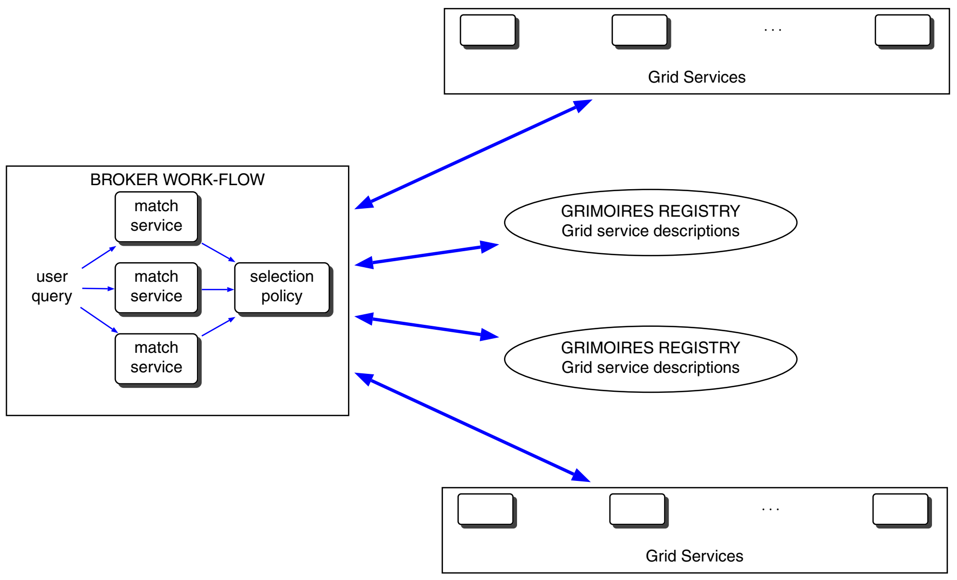


Figure 1. A high-level view of the KNOOGLE architecture

3 Broker Architecture

An outline of the KNOOGLE brokerage architecture is given in Figure 1 and comprises the following three components:

1. **Repositories:** that contain all the necessary information about the services that are to be evaluated against the query,
2. **Match Services:** that implement the various mechanisms to be used to compute the proximity of the match between the query and a service,
3. **Selection Policy:** that implements the (multi-dimensional) ranking mechanism for determining the most appropriate service as a result of all the match information collected.

Initially, a user may choose to use pre-packaged brokerage services, where repositories, match services and even selection policy have been fixed *a priori* at broker deployment. However, later users may want greater flexibility and control and be prepared to specify part or all of these elements themselves, either in the query (see later) or by deploying their own bespoke brokers — that are web services themselves. Indeed, eventually users may also choose to author their own matchers and selection policies and then deploy them in their own workflows as well as contributing them to the wider community.

Use Case 1: Matchmaking with client selection A client program sends a query to the broker. The broker fetches all the available service descriptions from the repositories registered with the broker. It then invokes each matcher on the query and each service description resulting in a set of (RDF) triples associating query, service and match-value. Finally, the set of triples — including the service details — is returned to the client for interpretation directly by the client application.

Use Case 2: Brokerage In this case, a client delegates service selection via a policy statement. This proceeds essentially as above except that the candidate set of services is then analyzed according to the client-specified policy and one service is selected and invoked.

4 Broker Operation and Selection Policy

At its most primitive, the broker is in fact just a skeleton, in which there is no repository, no match service and no selection policy. For it to do anything useful it must be supplied with at least one repository (this is assumed to be a UDDI-compliant repository: in our case we use the Grimoires [17] system), at least one matcher and a selection policy. While these would normally be specified at broker deployment, it is also possible to control these attributes dynamically through the query document. Thus the query document comprises a sequence of repositories (given as URLs), a sequence of matchers (given as service endpoints), a selection policy and the actual query as described by the pseudo-schema in figure 2.

The broker itself is a multi-threaded web service that deploys a pre-defined number of (internal) threads to process incoming queries, with further queries being held on

```
<querySubmission>
   <!-- the query document -->
   <query>
   <repositories>?
   <!-- the repositories to be used -->
     <repository>* <!-- repository URL -->
   <matchers>?
     <!-- the matchers to be used -->
     <matcher>* <!-- match service URL -->
   <!-- the selection policy script to use -->
   <selectionPolicy>?
     <!-- selection policy script language -->
     <selectionPolicyLanguage>?
```

Figure 2. A pseudo schema for the broker input document

a queue until a thread becomes available. The system is currently limited to a request-response model for the invocation of matchers because that is what Axis 1.0 supports. RDF triples are used to represent the relationship between a matcher and a service. These triples are inserted into a triple store — currently using the Sesame system [21] — while the selection policy may be stated in any of the three query languages supported by Sesame: RDQL (Resource Description Query Language), RQL (Resource Query Language) or SeRQL (Sesame RQL). In this way, we obtain a general-purpose representation of the match score data and a high-level language for specifying queries over the data, for example we give a query (in SeRQL) that selects the service that got the highest match score:

```
CONSTRUCT {serviceEndpoint} matcher {highestValue}
FROM
{serviceEndpoint} matcher {highestValue}
WHERE
highestValue >= ALL (
SELECT value
        FROM {} matcher {value})
```

As another example, we show how to implement a blacklist as part of the selection policy:

```
SELECT serviceEndpoint
FROM {serviceEndpoint} matcher
{highestValue}
WHERE highestValue >= ALL (
        SELECT value
        FROM {} matcher {value})
AND NOT (matcher LIKE "*Matcher2")
```

And finally (in RDQL), we show how to make a selection involving a numerical comparison of the two match scores, where the policy weights the results of `basicMatcher2` by a factor of three.

```
SELECT ?serviceEndpoint2
WHERE (?serviceEndpoint1
<http://localhost:8080/axis/services/Matcher2>
  ? value1),
(?serviceEndpoint2
<http://alis:9050/axis/services/basicMatcher2>
?value2) AND ?value1 < ?value2 * 3
```

While these examples are artificial, they demonstrate both the power and the flexibility available to broker clients and to brokerage service providers. Furthermore, they suggest how a policy may be stated that combines match score information from several matchers to identify a preferred service.

5 Broker Usage

Since the broker is a web service, interaction is mediated by web service invocation. However, clients also have the option of a command-line interface where a simple wrapper program takes a broker command followed by arguments and options. Alternatively, the broker may be used as a service within a workflow engine such as Taverna (see section 5.1). In the instance of command-line invocation the arguments are typically: (i) a URL identifying the broker itself, (ii) a query string (iii) a selection policy (iv) the selection policy language (RDQL, RQL or SeRQL) identifier (v) a matcher endpoint and (vi) a repository. The broker functions as a request/response service, so the outcome of this invocation is a notification. This key may subsequently be used by the client to poll the broker for the status of the match request as well as asking for more detailed information about the match process such as the set of match results received prior to the application of the selection policy.

5.1 Brokerage in Workflow Enactment

The Taverna workbench is a tool for creating and editing Simple Conceptual Unified Flow Language (SCUFL) scripts, that define workflows as a network of processors and links. SCUFL itself is interpreted by the Freefluo workflow enactment engine. The SCUFL language is primarily aimed at users who currently use web forms or scripting languages to interact with web resources [18]. The broker described above has been integrated as a service within the Taverna workflow, primarily by providing the broker as a standard Taverna service which then acts as a proxy for a particular service instance.

The novelty of this mechanism is that it now becomes possible to construct a form of abstract workflow. That is to say, conventional workflows in Taverna comprise concrete service endpoints that fulfill a particular task and have been identified and fixed at workflow design time. With the introduction of the broker described here, a workflow may now also contain broker instances that can be supplied with a query at the time of workflow enactment, thus deferring actual service identification until the time it is needed. While such flexibility may not always be necessary or desirable it means that: 1. workflow re-use need not depend on specific web services, but on a web service that satisfies a given task specification and 2. workflows can be constructed that determine the appropriate service at the time of need, which may be dependent on results established earlier in the workflow — a behaviour that was not previously possible.

Integration with Taverna is demonstrated in the context of GridSAM [19]: a system for managing the allocation of jobs to a pool of processors. The GridSAM system takes as input job descriptions in JSDL [16] and client-provided executables. It then distributes the jobs over the available processors that perform the processing, GridSAM then returns the results to the clients. GridSAM is conceived as a system in which clients know precisely which services they wish to invoke. Thus, the KNOOGLE broker provides an important complementary function such that clients may specify service *properties* that can be used to identify suitable GridSAM instances, given an appropriate match service. Using Taverna, we have developed a scenario to support job submission using the KNOOGLE broker and a special-purpose GridSAM matcher: see figure 3.

To demonstrate the generality of the approach, the broker has also been used for service discovery based on the Web Services Description Language (WSDL) – as illustrated in figure 3. In the screenshot of the Taverna window there are three panes: **top left:** displays the details of the workflow components comprising the workflow in the bottom pane **top right:** displays the list of available services that Taverna has found for the client and **bottom:** renders the workflow as a graph and permits the client to connect components one to another. On the left, the Knoogle broker is an element of the workflow that will be activated when control reaches this component, initiating the search for a suitable service and its subsequent invocation. On the right, the user specifies a service to discover, using terms in the WSDL schema – such as `OperationName` or `PortType`. This is then used as a basis to search through a UDDI registry service to find services of interest. As illustrated in figure, an `OperationName` and the location of the input data to pass to the service is submitted to the Broker, which searches through a collection of registries to undertake a syntax match on the advertised `OperationName`. The Broker returns the URL pointing to the location of the service and any other output information that can be found in the registry.

6 Related Work

Matchmaking has quite a significant body of associated literature. A broad categorizations of matchmaking and brokerage research seem possible using criteria such as domain, reasoning mechanisms and adaptability. Much of the published literature has described generic brokerage mechanisms using syntactic or semantic, or a combination of both, techniques. Some of the earliest systems, enabled by the development of KIF (Knowledge Interchange Format) [4] and KQML (Knowledge Query and Manipulation Language) [13], are SHADE [6] operating over logic-based and structured text languages and the complementary COINS [6] that operates over free text using well-known term-first index-first information retrieval techniques. Subsequent developments such as InfoSleuth [9] applied reasoning technology to the advertised syntax and semantics of a service description, while the RETSINA system [12] had its own specialized language influenced by DAML-S (the pre-cursor to OWL-S) and used a belief-weighted associative network representation of the relationships between ontological concepts as a central element of the matching process. While technically sophisticated, a particular problem with the latter was how to make the initial assignment of weights without biasing the system inappropriately. A distinguishing feature of all these systems is their monolithic architecture, in sharp contrast to GRAPPA [14] (Generic Request Architecture for Passive Provider Agents) which allows for the use of multiple matchmaking mechanisms. Otherwise GRAPPA essentially employs fairly conventional multi-attribute clustering technology to reduce attribute vectors to a single value. Finally, a notable contribution is the MathBroker architecture, that like the domain-specific plug-ins of our brokerage scheme, works with semantic descriptions of mathematical services using the same MSDL language. However, current publications [1] seem to indicate that matching is limited to processing taxonomies and the functional issues raised by pre- and post-conditions are not considered. The MONET broker [3], in conjunction with the RACER reasoner and the Instance Store demonstrated one of the earliest uses of a description logic reasoner to identify services based on taxonomic descriptions coming closest to the objective of the plug-ins developed for GENSS in attempting to provide functional matching of task and capability.

In contrast, matching and brokerage in the Grid computing domain has been relatively unsophisticated, primarily using syntactic techniques, such as in the ClassAds system [11] used in the Condor system and RedLine [7] which extends ClassAds, where match criteria may be expressed as ranges and hence are a simple constraint language. In the Condor system, the use of ClassAds is to enable computational jobs find suitable resources, generally using dynamic attributes such as available physical and virtual memory, CPU type and speed, current load average, and other static attributes such as operating system type, job manager etc. A resource also has a simple policy associated with it, which identifies when it is willing to accept new job requests. The approach is therefore particular focused to work for managing job execution on a Condor pool, and configured for such a system only. It would be difficult to deploy this approach (without significant changes) within another job execution system, or one that makes use of a different resource model. The RedLine system allows matching of job requests with

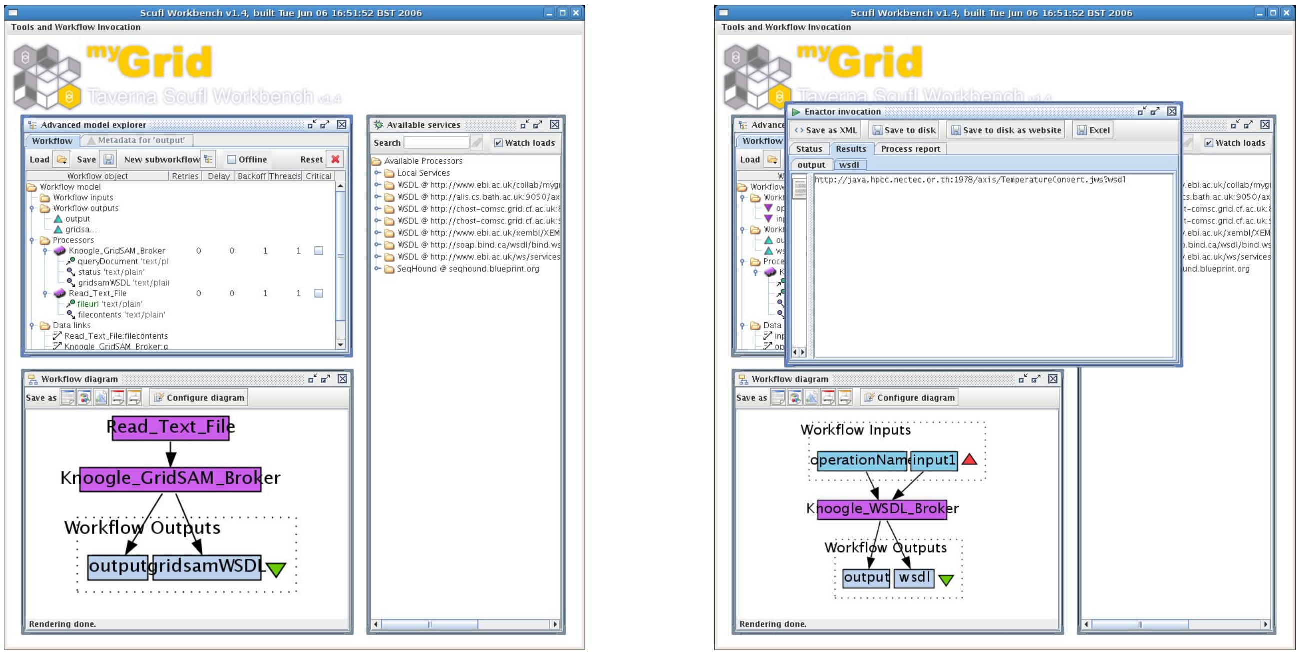


Figure 3. Integrating GridSAM with Taverna workflow enactment (left) and A Taverna workflow using a discovered Web Service using its WSDL description (right)

resource capabilities based on constraints – in particular the ability to also search based on resource policy (i.e. when a resource is able to accept new jobs, in addition to job and resource attributes). The RedLine description language provides functions such as *Forany* and *Forall* to be able to find multiple items that match. The RedLine system is however still constrained by the type of match mechanisms that it supports—provided through its description language. Similar to Condor, it is also very difficult to modify it for a different resource model. Our approach is more general, and can allow plug-ins to be provided for both RedLine and Condor as part of the matchmaker configuration. In our model, therefore, as the resource model is late-bound, we can specify a specialist resource model and allow multiple such models to co-exist, each implemented in a different configuration of the matchmaker.

7 Performance indicators

Performance evaluation is necessarily dominated by network latency and overheads and so we consider immaterial the details of individual machines involved. Some indicative tests have been carried out over the Joint Academic NETwork (JANET) infrastructure in the UK, where the main inter-university links are 2Mb/s. Performance indicators are based on an "end-to-end" test, with a collection of combinations of client, matchers and registries, both local and remote. Not surprisingly, delays are quite significant given the general-purpose network being used, such that with a client in Bath and a registry in Cardiff, we observed wall-clock round-trip time of 6–8 seconds. The results presented here apply equally to workflows, in that the same overheads will apply when the broker is embedded in a workflow as when it is run stand-alone. A soak test in which we repeatedly ran the same search (client/matching process – as a Web Service – on a machine in Bath with registry in Cardiff) produced the following results:

Structural matcher

Iterations	real	user	sys
1	0m14.495s	0m4.362s	0m0.172s
10	1m2.898s	0m5.766s	0m0.318s
100	9m59.015s	0m12.173s	0m1.566s

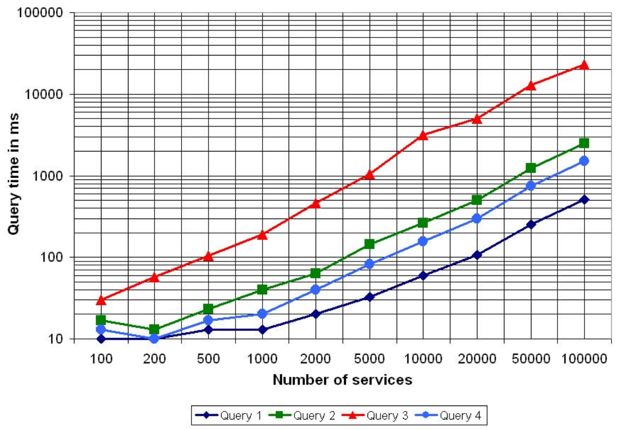
Indicating that amortized elapsed time/query is around 6 seconds using the structural matcher. A further test using a different matcher produced the following results:

Ontological matcher

Iterations	real	user	sys
1	0m22.089s	0m4.786s	0m0.173s
10	1m49.031s	0m5.882s	0m0.299s
100	17m13.894s	0m12.682s	0m1.203s

Indicating that amortized elapsed time/query is just over 10 seconds using the ontological matcher. Performing the same test as above, but where a command line Java application was used to search the service registry, Linux system monitoring tools report that on the machine carrying out the search process that program used approximately 25.4MB and 36.06% CPU.

To evaluate the cost of using reasoning in the matchmaking chain, we have measured the performance of the OWL-JessKB rule engine using the ontologies developed in the MONET project [10]. The MONET ontologies consist of 2031 classes, 78 slots and 10 facets. This leads to a heap size of 109Mb. As expected there is a linear relationship between the size of the ontology and the heap size. However, query costs are essentially flat with respect to ontology size. More importantly, the scalability with respect to the number of services is linear: measurements were taken populating the registry with an increasing number of services, while using the above MONET ontology, operating the JVM in a maximum heap size of 1 GB. The registry was populated with differing numbers of services (see below) and query performance is linear from a level in the noise of 10-20ms up to 500 services up to between 2000-20000ms for 100,000 services, depending on the type of query:



The query types were:

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 by a single object
4. Assertion aggregate: Confirm if constraint y is satisfied for a group of objects.

This indicate that for foreseeable registry sizes over a range of queries, rule-based reasoning should not constitute a major part of the cost.

8 Conclusion

The requirements for a generic, re-targettable matchmaker and broker have been set out and a novel architecture that satisfies them has been outlined. We have built such a broker that may be populated by a mixture of generic and domain-specific plug-ins. Furthermore, these plug-ins may also be composed and deployed with low overhead, especially with the help of workflow tools, to create bespoke matchmakers/brokers. The plug-ins are implemented as web services and a mechanism has been provided to integrate them into the architecture. This provide a basis from which to explore the definition of policies for the combination of results from multiple match techniques. The broker has been integrated as a proxy service within the Taverna workflow engine, enabling the construction of a form of abstract workflow. By separating broker knowledge from broker operation, we hope this will enable developers to concentrate on the creation of rich service descriptions and the matchers that can take advantage of that knowledge, while ensuring such efforts can readily be deployed and re-used, by building on the web services infrastructure.

9 Acknowledgments

In its initial phases, the work reported here was partially supported by the Engineering and Physical Sciences Research Council (UK) under the Semantic Grids call of the e-Science program (grant reference GR/S44723/01). The development of the broker middleware itself has subsequently been supported by the Open Middleware Infrastructure Institute managed program as project KNOOGLE. The software is currently undergoing integration with the OMII client/server distribution and should form part of a forthcoming release (BSD licence).

References

[1] Rebhi Baraka, Olga Caprotti, and Wolfgang Schreiner, "A Web Registry for Publishing and Discovering Mathematical Services." In *EEE*, pages 190–193. IEEE Computer Society, 2005.

[2] Olga Caprotti, Michael Dewar, James Davenport, and Julian Padget. Mathematics on the (Semantic) Net. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *Proceedings of the European Symposium on the Semantic Web*, volume 3053 of *LNCS*, pages 213–224. Springer Verlag, 2004. ISBN 3-540-21999-4.

[3] Olga Caprotti, Mike Dewar, and Daniele Turi, "Mathematical Service Matching Using Description Logic and OWL.", In Andrea Asperti, Grzegorz Bancerek, and Andrzej Trybulec, editors, *MKM*, volume 3119 of *Lecture Notes in Computer Science*, pages 73–87. Springer, 2004.

[4] M. Genesereth and R. Fikes, “Knowledge Interchange Format, Version 3.0 Reference Manual”. Technical report, Computer Science Department, Stanford University, 1992. Available from `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`.

[5] Duncan Hull, Robert Stevens, Phillip Lord, Chris Wroe, and Carole Goble. “Treating ‘shimantic web’ syndrome with ontologies”. *First Advanced Knowledge Technologies workshop on Semantic Web Services (AKT-SWS04)*, volume 122. KMi, The Open University, Milton Keynes, UK, 2004.

[6] D. Kuokka and L. Harada, “Integrating information *via* matchmaking”. *Intelligent Information Systems 6(2-3), pp. 261-279*, 1996.

[7] Chuang Liu and Ian T. Foster. A Constraint Language Approach to Matchmaking. In *RIDE*, pages 7–14. IEEE Computer Society, 2004.

[8] Simone Ludwig, Omer Rana, William Naylor, and Julian Padget. Matchmaking Framework for Mathematical Web Services. *Journal of Grid Computing*, 4(1):33–48, March 2006. Available via `http://dx.doi.org/10.1007/s10723-005-9019-z`. ISSN: 1570-7873 (Paper) 1572-9814 (Online).

[9] W. Bohrer M. Nodine and A.H. Ngu. Semantic brokering over dynamic heterogenous data sources in InfoSleuth. In *Proceedings of the 15th International Conference on Data Engineering, pp. 358-365*, 1999.

[10] Mathematics on the Net - MONET. `http://monet.nag.co.uk`.

[11] Rajesh Raman, Miron Livny, and Marvin H. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *HPDC*, pages 140–, 1998.

[12] Katia P. Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph A. Giampapa. The RETSINA MAS Infrastructure. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):29–48, 2003.

[13] D. McKay T. Finin, R. Fritzson and R. McEntire, “KQML as an agent communication language”. *Proceedings of 3rd International Conference on Information and Knowledge Management, pp. 456-463*, 1994.

[14] D. Veit, “Matchmaking in Electronic Markets”, LNCS2882, Springer, 2003. Hot Topics.

[15] Amy Moormann Zaremski and Jeannette M. Wing, “Specification Matching of Software Components”. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, October 1997.

[16] Job Submission Description Language Specification v1.0. Available at: `http://www.gridforum.org/documents/GFD.56.pdf`. Last accessed: February 2007.

[17] Luc Moreau, “GRIMOIRES: Grid RegIstry with Metadata Oriented Interface: Robustness, Efficiency, Security”. `http://twiki.grimoires.org/bin/view/Grimoires/`. Last accessed: January 2007.

[18] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Anil Wipat, Peter Li and Tim Carver, “Delivering Web Service Coordination Capability to Users. *Proceeding of the WWW conference, 2004, May 17–22, New York, USA.*

[19] GridSAM - Grid Job Submission and Monitoring Web Service. Available at: `http://gridsam.sourceforge.net/2.0.0/`. Last accessed: February 2007.

[20] Tom Goodale, Simone Ludwig, William Naylor, Julian Padget, Omer Rana, “Service-Orientated Matchmaking and Brokerage”, Proceedings of AHM 2006, Nottingham, UK, Aug./Sept. 2006.

[21] Sesame: RDF Schema Querying and Storage. Available at: `www.openrdf.org/`. Last accessed: February 2007.

[22] KNOOGLE: Matchmaking and Brokerage Framework. OMII Project: 2006–2007. `http://www.omii.ac.uk/downloads/project.jsp?projectid=77`. Last accessed: September 2007.

[23] The Open Middleware Infrastructure Institute (OMII-UK) program. `http://www.omii.ac.uk`. Last accessed: September 2007.