SCALABLE COMPUTING: PRACTICE AND EXPERIENCE
Volume 7, Number 2, pp. 35–51. http://www.scpe.org

ISSN 1895-1767
© 2006 SWPS

MATHEMATICAL SERVICE DISCOVERY: ARCHITECTURE, IMPLEMENTATION AND PERFORMANCE

SIMONE A. LUDWIG*, OMER F. RANA†, WILLIAM NAYLOR‡, AND JULIAN PADGET§

Abstract. Service discovery and matchmaking in a distributed environment has been an active research issue since at least the mid 1990s. Previous work on matchmaking has typically presented the problem and service descriptions as free or structured (marked-up) text, so that keyword searches, tree-matching or simple constraint solving are sufficient to identify matches. In this paper, we discuss the problem of matchmaking for mathematical services, where the semantics play a critical role in determining the applicability or otherwise of a service. A matchmaking architecture supporting the use of match plug-ins is first described, followed by the types of plug-ins that can be supported. The matched services are ranked based on the score obtained from each plug-in, with the user being able to decide which plug-in is most significant in the context of their particular application. We consider the effect of pre- and post-conditions of mathematical service descriptions on matching, and how and why to reduce queries into DNF and CNF before matching. Application examples demonstrate in detail how the matching process works for all four algorithms. Additionally, an evaluation of the ontological mode is provided, regarding performance of loading ontologies, query response time and the overall scalability is conducted. The performance results are used to demonstrate scalability issues in supporting ontology-based discovery within a Web Services environment.

Key words. mathematical Web Services, matchmaking, match score, performance, scalability.

1. Introduction.

The amount of machine-oriented data on the Web is increasing rapidly as semantic Web technologies achieve greater up-take. Humans typically use Google to search for suitable services, but they can filter out the irrelevant and spot the useful. Therefore, while UDDI (the Web Services registry) with keyword searching essentially offers something similar, it is a long way from being very helpful. Often, it is difficult to formulate a query in a precise way, therefore resulting in the results of search engines not being suitable for automated services—as many of the matches returned by the search may not be directly relevant to the query. A human user is therefore necessary to evaluate the outcome of a search, and make manual selection between the set of results that have been returned. Conversely, when using automated registry services, the options for specifying a query are limited, therefore resulting in a service that has limited usability.

To address this concern, much research has taken place on intelligent brokerage, such as Infosleuth [2], LARKS [3], and IBROW [4]. It is perhaps telling that much of the literature appears to focus on architectures for brokerage, which are as such domain-independent, rather than concrete or domain-specific techniques for identifying matches between a *task* or problem description and a *capability* or service description. Approaches to matching in the literature fall into two broad categories:

- *syntactic matching*, such as textual comparison or the presence of keywords in free text. In these approaches, the terms used in the query are matched with those in the service description. Either the existence (or not) of terms is used to undertake a comparison, or any attributes or data values associated with these terms are also compared.
- *semantic matching*, which typically seems to mean finding elements in structured (marked-up) data and perhaps additionally the satisfaction of constraints specifying ranges of values or relationships between one element and another.

For many problems this is both appropriate and adequate, indeed it is not clear what more one could do, but in the particular domain of mathematical services the actual mathematical semantics are critical to determining the suitability (or otherwise) of the capability for the task. The requirements are neatly captured in [5] by the following condition:

$$T_{in} \geq C_{in} \wedge T_{out} \leq C_{out} \wedge T_{pre} \Rightarrow C_{pre} \wedge C_{post} \Rightarrow T_{post} \tag{1.1}$$

where T refers to the task, C to the capability, *in* are inputs, *out* are outputs, *pre* are pre-conditions and *post* are post-conditions. What the condition expresses is that the signature constrains the task inputs to be at least as great as the capability inputs (i. e. enough information), that the inverse relationship holds for the

*School of Computer Science, Cardiff University, UK (Simone.Ludwig@cs.cardiff.ac.uk)
†School of Computer Science, Cardiff University, UK (O.F.Rana@cs.cardiff.ac.uk)
‡Department of Computer Science, University of Bath, UK (wn@bath.ac.uk)
§Department of Computer Science, University of Bath, UK (jap@bath.ac.uk)

outputs and there is a pairwise implication between the respective pre- and post-conditions. This however leaves unaddressed the matter of establishing the validity of that implication.

In the MONET [6, 8] and GENSS [9] projects the objective is mathematical problem solving through service discovery and composition by means of intelligent brokerage. *Mathematical* capability descriptions turn out to be both a blessing and a curse: precise service description are possible thanks to the use of the OpenMath [10] mathematical semantic mark-up, but service matching can rapidly turn into intractable (symbolic) mathematical calculations unless care is taken.

As Grid computing adopts the Service Oriented Architecture for service usage and deployment, a matchmaker can be seen as another infrastructure service—deployed as part of a discovery infrastructure. Many of the capabilities of a "matchmaker" are similar to those of a "broker", as adopted in many existing Grid computing applications. However, a key distinction we make is that a broker does not provide any support for scheduling or executing a service once it has been discovered. The focus here is on numerical services, as they are generally much better understood, and therefore may also serve as useful benchmarks for evaluating a system. Furthermore, multiple instances of mathematical services (implemented by a variety of different vendors) can be found, thereby providing the capability to choose between similar services made available over different deployment platforms.

This paper discusses our experience with developing such a domain-specific matchmaking approach for mathematical services. We suggest the use of a different matchmaking mechanisms to answering the implication question posed above. The set of ontologies provided by the MONET project have been used to specify a query. This query is subsequently used to undertake a search against the services that have been registered in a registry. Performance issues are also discussed. In the next section we discuss the description of mathematical services, in section 3 we describe the web services-based matchmaking architecture and detail the roles of the components within it, in section 4 we provide application examples outlining integer factorisation. Performance results are presented in section 5, and in section 6 we summarise and conclude this paper.

2. Description of Mathematical Services. The OpenMath [10] has been introduced as a means to describe properties of mathematical objects (for exchange between computer programs). OpenMath is a mark up language for representing the semantics (as opposed to the presentation) of mathematical objects in an unambiguous way. It may be expressed using an XML syntax. OpenMath expressions are composed of a small number of primitives. The definition of these may be found in [10], for instance: `OMA` (OpenMath Application), `OMI` (OpenMath Integer), `OMS` (OpenMath Symbol) and `OMV` (OpenMath Variable). Symbols are used to represent objects defined in the Content Dictionaries (to be discussed). Applications specify that the first child is a function or operator to be applied to the following children. As an example, the expression $x + 1$ might look like:[1]

```
<om:OMA>
  <om:OMS cd="arith1" name="plus"/>
  <om:OMV name="x"/>
  <om:OMI> 1 </om:OMI>
</om:OMA>
```

where the symbol `plus` is defined in the *Content Dictionary* (CD) `arith1`. Content Dictionaries are definition repositories in the form of files defining a collection of related symbols and their meanings, together with various *Commented Mathematical Properties* (for human consumption) and *Formal Mathematical Properties* (for machine consumption). The symbols may be uniquely referenced by the CD name and symbol name via the attributes `cd` and `name` respectively, as in the above example. Another way of thinking of a CD is as a small, specialised ontology.

MathML comes in two forms:

- for presentation (rendering in browsers) and
- for content (semantics),

and both are W3C recommendations. The specification ([11] section 5.1) and [12] identify ambiguities in presentation MathML. Content MathML is designed to handle the semantics of a limited subset of mathematics up to *K-12* level, mathematics beyond this may be encoded by using OpenMath and the *semantics* tag, alternatively *parallel markup* may be utilised [11]. There are various ways in which OpenMath can help in matchmaking:

[1]Throughout the paper, the prefix `om` is used to denote the namespace: `http://www.openmath.org/OpenMath`

- OpenMath can be used to encode the mathematical part of a problem to be solved in a query, for example a differential equation or an integral.
- OpenMath may be used to encode the input parameters to be sent to a service and the values returned by the service.
- The function of a service (together with the signature of the input parameters, and output objects) may be described in OpenMath, these may then be encoded using specialist tags to form a mathematical service description; described in Mathematical Service Description Language (MSDL) [13].

MSDL is an extension of WSDL that was developed as part of the MONET project [6], incorporating more information about a service, in particular pre- and post-conditions, taxonomic references etc.

3. Mathematical Matchmaking. Matchmaking allows potential producers of information to advertise their capability, and subsequently consumers of information to send messages describing their information needs. These descriptions, represented in rich, machine-interpretable description languages, are unified by the matchmaker to identify potential matches [7]. A Web Services-based matchmaking architecture is presented, describing the role played by different components within the architecture. Also described are the schemas and ontologies that may be used as part of the producer advertisement or consumer requirement.

3.1. Schemas and Ontologies. The XML document schemas we are using in GENSS are, at the moment, those developed for the MONET project. There are three main schemas:

- Mathematical Service Description Language (MSDL)
- Mathematical Problem Description Language (MPDL) and
- Mathematical Query Description Language (MQDL)

whose purposes are apparent in the second word in each case. The obvious question, even criticism, is why develop this range of languages when there is DAML-S [14] and OWL-S [15]? The answer is purely practical: at the time of the MONET project (April 2002 to March 2004) OWL and OWL-S were still subject to change and there were hardly any tools available, while DAML and DAML-S were clearly about to be made obsolete by OWL/OWL-S and the tool situation was hardly any better. Thus a pragmatic decision was made to take the principles needed to enable the MONET deliverables from DAML/OWL and embed them in some simple restricted languages over which the project had full control. Thus we see the adoption of pre- and post-condition driven descriptions of capabilities and tasks, following the ideas set out in DAML/OWL and being explored in various semantic brokerage projects such as InfoSleuth [2], RETSINA [3] and IBROW [16], while embedding WSDL in MSDL to provide the necessary information about how to invoke the service. It is our intention to explore how we can move from MSDL towards OWL/OWL-S during the lifetime of the current GENSS project, since this will greatly aid interoperability and enable the utilisation of the increasing range of tools (for OWL/OWL-S) that have become available.

History is also the explanation for the ontology language we use. OpenMath [10] provided an early use of XML for providing an application-specific description—before the availability of RDF. Nevertheless, OpenMath stands as probably the most developed ontology of mathematics, because in contrast to MATHML-C [11] it is extensible through the mechanism of content dictionaries which were developed to handle the absence of modularisation facilities or namespaces in the original XML. The OpenMath 2 standard [18] replaces DTDs with schemas, provides compatibility with RDF tools, utilises XML namespaces and generally aims to bring OpenMath in line with developments in ontology representation over the last five years, whilst keeping where feasible, backwards compatibility with OpenMath 1.1. Thus we make use of OpenMath as the primary representation language for mathematical content in our work.

3.2. Related Matchmaking Approaches. A variety of matchmaking systems have been reported in literature, we review some related systems below.

The SHADE (SHAred Dependency Engineering) matchmaker [19] operates over logic-based and structured text languages. The aim is to dynamically connect information sources. The matchmaking process is based on KQML (Knowledge Query and Manipulation Language) communication [20]. Content languages of SHADE are a subset of KIF (Knowledge Interchange Format) [21] as well as a structured logic representation called MAX (Meta-reasoning Architecture for "X"). Matchmaking is carried out solely by matching the content of advertisements and requests. There is no knowledge base and no inference performed. The SHADE matchmaker is implemented entirely as a forward-chaining rule-based program using MAX. This allows adding features such as new rules dynamically.

COINS (COmmon INterest Seeker) [19] is a matchmaker which operates over free text. The motivation for the COINS is the need for matchmaking over large volumes of unstructured text on the Web or other Wide Area Networks and the impracticality of using traditional matchmakers in such an application domain. Initially the free text matchmaker was implemented as the central part of the COINS system but it turned out that it was also useful as a general purpose facility. As in SHADE the access language is KQML. The System for the Mechanical Analysis and Retrieval of Text (SMART) [22] information retrieval system is used to process free text. The text is converted into a document vector using SMART's stemming and "noise" word removal. Then the document vectors are compared using an inverse document frequency algorithm.

LARKS (Language for Advertisement and Request for Knowledge Sharing) [3] was developed to enable interoperability between heterogeneous software agents and had a strong influence on the DAML-S specification. The system uses ontologies defined by a concept language ITL (Information Terminology Language). The technique used to calculate the similarity of ontological concepts involves the construction of a weighted associative network, where the weights indicate the belief in relationships. While it is argued that the weights can be set automatically by default, it is clear that the construction of realistically weighted relationships requires human involvement, which becomes a hard task when thousands of agents are available.

InfoSleuth [2] is a system for discovery and retrieval of information in open and dynamically changing environments. The brokering function provides reasoning over the advertised syntax and the semantics. InfoSleuth aims to support cooperation among several software agents for information discovery, where agents have roles as core, resource or ontology agents. A central service is the broker agent which is equipped with a matchmaker which matches agents that require services with agents that can provide those services. To apply this procedure an advertising agent has to register with the broker agent. The broker inserts the agent's description into its broker repository. The broker can then execute queries by requesting agents. These queries are formulated by agents who need other agents to fulfil their tasks.

The GRAPPA [23] (Generic Request Architecture) system allows multiple types of matchmaking mechanisms to be employed within a system. It is based on receiving arbitrary matchmaking offers and requests, where each offer and request consist of multiple criteria. Matching is achieved by applying distance functions which compute the similarities between the individual dimensions of an offer and a request. Using particular aggregate functions, the similarities are condensed to a single value and reported to the user.

The MathBroker (and MathBroker II) project(s) at RISC-Linz have some elements in common with those described here, including providing semantic descriptions of mathematical services. They too use MSDL, however it seems that most of the matchmaking is achieved through traversing taxonomies, while actual reasoning about the pre- and post-conditions is still an open problem.

Most of the projects above have focused on providing a generic matchmaker, capable of being adapted for a particular application. However, the motivation for many such projects has primarily been e-commerce (as a means to match buyers with sellers, for instance). Some projects are also focused on the use of a particular multi-agent interaction language (such as KQML), to enable communication between the matchmaker and other agents. Our approach, however, is centered on the implementation of a matchmaker that is specific to mathematical relations. Similar to GRAPPA, our matchmaker can support multiple comparison techniques.

3.3. Matchmaking Requirements. To achieve matchmaking:

- we want sufficient input information in the task to satisfy the capability, while the outputs of the matched service should contain at least as much information as the task is seeking, and
- the task pre-conditions should be more than satisfied by the capability pre-conditions, while the post-conditions of the capability should be more than satisfied by the post-conditions of the task.

These constraints reflect work in component-based software engineering and are, in fact, derived from [24]. They are also more restrictive than is necessary for our setting, by which we mean that some inputs required by a capability can readily be inferred from the task, such as the lower limit on a numerical integration or the dependent variable in a symbolic integration. Conversely, a numerical integration routine might only work from 0 to the upper limit, while the lower limit of the problem is non-zero. A capability that matches the task can be synthesised from the composition of two invocations of the capability with the fixed lower limit of 0. Clearly the nature of the second solution is quite different from the first, but both serve to illustrate the complexity of this domain. It is precisely this richness too that dictates the nature of the matchmaking architecture, because as these two simple examples show, very different reasoning capabilities are required to resolve the first and the second. Furthermore, we believe that given the na-

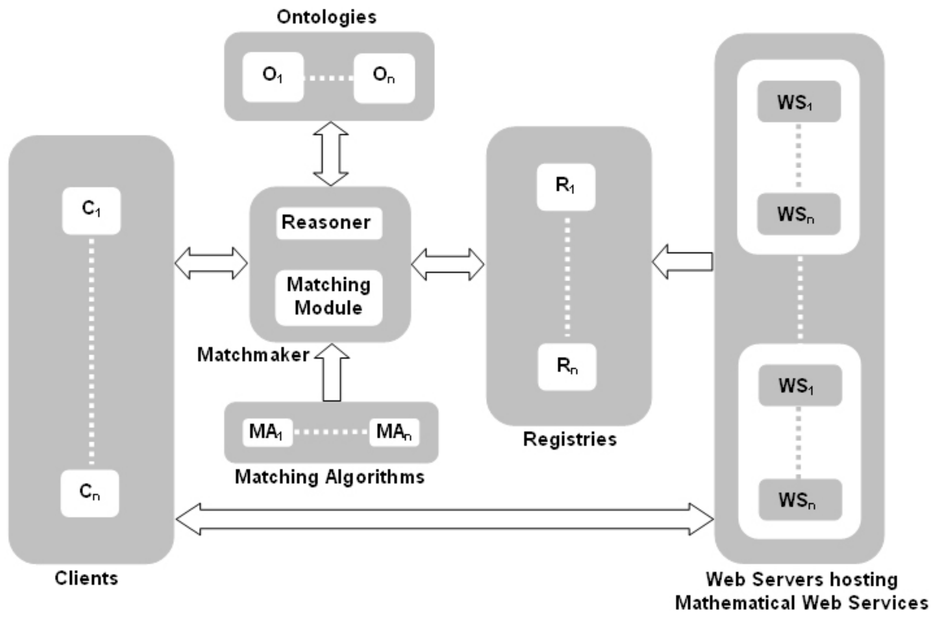

Fig. 3.1. *Matchmaking Architecture*

ture 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. This results in:

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

The second problem is a consequence of the above: there will potentially be several candidate matches and some means of indicating their suitability is desirable, rather than picking the first or choosing randomly. Thus:

Requirement 2: A ranking mechanism is required that takes into account pure technical (as discussed above in terms of signatures and pre- and post-condition) and quantitative and qualitative aspects—and even user preferences.

3.4. Matchmaking Architecture. Our matchmaking architecture is outlined in Figure 3.1 and comprises the following parts:

- Client interface: which may be employed by the user to specify their service request. The client interface may be a Web browser, or may be a special purpose interface that has been embedded within an existing application.
- Matchmaker: which contains a reasoning engine and the matching module. The reasoning engine relies on the existence of one or more domain specific ontologies.
- Matching algorithm Web Services: where the logic of the matching mechanism is defined. These Web Services primarily provide the plug-in capability that has been identified in Requirement 1 above. Currently, there is no mechanism to combine multiple match algorithms. Each algorithm therefore must be used independently.
- Mathematical ontologies: such as OpenMath CDs, GAMS etc. These ontologies need to be used alongside the matchmaker and the reasoner.
- Registry service: where the mathematical service descriptions are stored. A registry in this context contains the set of "advertisement" (or metadata) that define numerical algorithms that have been made available for use. In our architecture, we assume that these algorithms have been implemented as Web Services. The registry does not contain any executables, only references to executables that are maintained elsewhere.
- Mathematical Web Services: available on third party sites, accessible over the Web. These are the real executables associated with descriptions that are maintained in the registry.

The interactions of a search request are as follows:

- a user contacts the matchmaker, then
- the matchmaker loads the matching algorithms specified by the user; in the case of an ontological match, further steps are necessary;
- the matchmaker contacts the reasoner which in turn loads the corresponding ontology;
- having additional match values results in the registry being queried, to see whether it contains services which match the request and finally
- service details are returned to the user via the matchmaker. These details generally include an end point reference for the service that is being maintained on a remote file system.

The parameters stored in the registry (a database) are service name, URL/end point reference, taxonomy/ontology, input, output, pre- and post-conditions. Using contact details of the service from the registry, the user can then call the Web Service and interact with it. Each component of the architecture is now described in more detail.

3.4.1. Matching Algorithms. Currently four matching algorithms have been implemented within the matchmaker:

- structural match;
- syntax and ontological match;
- algebraic equivalence match;
- value substitution match.

These matchers are complementary and constitute the polyalgorithmic approach mentioned in the abstract. The structural match only compares the OpenMath symbol element structures (e.g. `OMA`, `OMS`, `OMV` etc.). The syntax and ontological match algorithm goes a step further and compares the OpenMath symbol elements and the content dictionary values of `OMS` elements. If a syntax match is found, which means that the values of the content dictionary are identical, then no ontology match is necessary. If an ontology match is required, the query structure is matched using the content dictionary hierarchy. The algebraic equivalence match and value substitution match do actual mathematical reasoning using the OpenMath structure.

The **structural match** works as follows: the pre- and post-conditions are extracted and an SQL query is constructed to find the same OpenMath structure of the pre- or post-conditions of the service descriptions in the database.

The **ontological match** is performed similarly, however, the OpenMath elements are compared with an ontology [25] representing the OpenMath elements. The matchmaking mechanism allows a more efficient matchmaking process by using mathematical ontologies such as the one for sets shown in Figure 3.2. OWL-JessKB [26] was used to implement the ontological match. It is intended to facilitate reading Ontology Web Language (OWL) files, interpreting the information as per OWL and RDF languages, and allowing the user to query on that information. To give an example the user query contains the OpenMath element:
`<om:OMS cd='setname1' name='Z'/>`
and the service description contains the OpenMath element:
`<om:OMS cd='setname1' name='P'/>`
which refer to the set of integers and the set of positive prime numbers respectively.

The query finds the entities `Z` and `P` and determines the similarity value depending on the distance between the two entities (inclusive, on one side) which in this case is $SV = \frac{1}{n} = 0.5$, where n is the number of nodes in the ontology that separate `Z` from `P`. As can be seen from figure 3.2 this is 2—the figure also identifies that concept `P` is subsumed by concept `Z`. Our use of distance in this context therefore differs from how it is used in general when evaluating similarity between concepts within an ontology. A detailed description of the match making algorithm can be found in [1].

For both the ontological and structural match, it is necessary that the pre- and post- conditions are in some standard form. For instance, consider the algebraic expression $x^2 - y^2$, this could be represented in OpenMath as:

```
<om:OMOBJ><om:OMA>
  <om:OMS cd="arith1" name="minus"/>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="x"/>
```

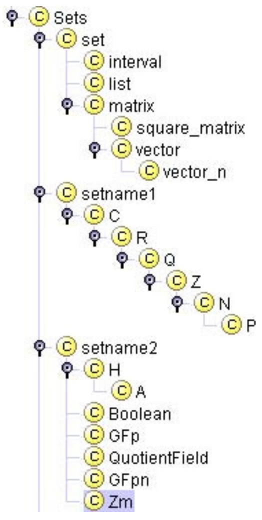

FIG. 3.2. *Set Ontology Fragment*

```
    <om:OMI>2</om:OMI>
  </om:OMA>
  <om:OMA>
    <om:OMS cd="arith1" name="power"/>
    <om:OMV name="y"/>
    <om:OMI>2</om:OMI>
  </om:OMA></om:OMA>
</om:OMOBJ>
```

however, $x^2 - y^2 = (x + y)(x - y)$, leading to the ontologically and structurally different markup:

```
<om:OMOBJ><om:OMA>
  <om:OMS cd="arith1" name="times"/>
  <om:OMA>
    <om:OMS cd="arith1" name="plus"/>
    <om:OMV name="x"/>
    <om:OMV name="y"/>
  </om:OMA>
  <om:OMA>
    <om:OMS cd="arith1" name="minus"/>
    <om:OMV name="x"/>
    <om:OMV name="y"/>
  </om:OMA></om:OMA>
</om:OMOBJ>
```

Both are “right”, it just depends on what information is wanted, so there can in general be no canonical form. So in order to address the above observation, we must look deeper into the mathematical structure of the expressions which make up the post-conditions. Most of the conditions examined may be expressed in the form: Q(L(R)) where:

- Q is a quantifier block, e.g. $\forall x \exists y$ s.t. $\cdots$
- L is a block of logical connectives, e.g. $\wedge, \vee, \Rightarrow, \cdots$
- R is a block of relations, e.g. $=, \leq, \geq, \neq, \cdots$

Processing the Quantifier Block. In most cases, the quantifier block will just be a range restriction. In other cases it may be possible to use *quantifier elimination* to replace the quantifier block by an augmented logical block. Quantifier elimination is a problem for which code exists in many computer algebra systems; e.g. *RedLog* in Reduce.

Processing the Logical Block. Once the quantifier elimination has been performed on the query descriptions and the service descriptions, the resulting logical blocks must be converted into normal forms. The logical block of both the service and query descriptions are converted to disjunctive normal form (DNF—that is a form which only contains a disjunction of conjunctions of terms and term negations). We need to now calculate a value which determines how well equation (1.1) is satisfied. That is to say, the pre-conditions of the service must be satisfied by the pre-conditions of the query and the post-conditions of the query must be satisfied by the post-conditions of the service. In all the following , we consider pre- and post- conditions in DNF, so $x \in C_{pre}$ means *x is a conjunct in the DNF for the service pre-condition.* Superfluous service pre-conditions (query post-conditions) do not effect whether the function may be performed. It is necessary, however, that there are no extra query pre-conditions (service post-conditions) as this might allow the client to provide conditions incompatible with the service pre-condition (service post-condition). This may be formalised in the following:

$$\forall x_1 \in T_{pre}\ \exists y_1 \in C_{pre} \ \text{ s.t. } \mathrm{x}_1 \Rightarrow \mathrm{y}_1 \tag{3.1}$$

and

$$\forall x_2 \in C_{post}\ \exists y_2 \in T_{post} \ \text{ s.t. } \mathrm{x}_2 \Rightarrow \mathrm{y}_2 \tag{3.2}$$

One way of proceeding is to treat the pre- and post- conditions separately in order to get two similarity values $\mathcal{S}_{pre}$ and $\mathcal{S}_{post}$. If it so happens that the pre- and post- conditions are equally important, then the average of these values will provide a good measure for the similarity value, however this will not always be the case, and other feasible measures are to weight $\mathcal{S}_{pre}$ and $\mathcal{S}_{post}$ linearly with the number of matching disjuncts in the pre-condition match as opposed to the post-condition match. This can be justified by observing that there are a linear number of different ways for the conditions to match.

We shall denote the DNF for C_{pre} (or T_{post}) by $R = R_1 \vee \cdots \vee R_n$ and for C_{post} (or T_{pre}) by $S = S_1 \vee \cdots \vee S_{\tilde{n}}$. To calculate a value $\in [0.0, 1.0]$ indicating how well equations 3.1, 3.2 are satisfied, we shall use the formula:

$$\mathit{similarity}(R, S) = \sum_{i=1..\tilde{n}} M_1(R, S_i)\frac{1}{\tilde{n}} \tag{3.3}$$

where M_1 is a function which indicates how well the expression $S_i \Rightarrow R$ holds. This is equivalent to stating how well S_i matches with one of the conjuncts making up R. A good formula to calculate this is:

$$M_1(R, S_i) = \max_{j=1\cdots n}\{M_2(R_j, S_i)\} \tag{3.4}$$

where M_2 is a similarity function for conjuncts. We may calculate a value for $M(R_i, S_j)$ as:

$$M_2(R_j, S_i) = \sum_{k=1\cdots\delta} m(R_j, S_{i,k})\frac{1}{\delta} \tag{3.5}$$

where δ is the number of terms in S_i, $S_{i,k}$ are terms in S_i and:

$$m(R_j, S_{i,k}) \text{ returns } \quad \begin{array}{l} 1.0 \text{ if } S_{i,k} \text{ matches a term in } R_j, \\ 0.0 \text{ otherwise.} \end{array} \tag{3.6}$$

Processing the Relations Block. In order to perform the term matches necessary to calculate 3.6 we shall consider two possible methods. It is useful to note that a term is of the general form: $T_L \succ T_R$ where $\succ$ is some relation i. e. a predicate on two arguments. In the case that T_L and T_R are real values, we may proceed as follows: we have two terms we wish to compare $Q_L \succ Q_R$ and $S_L \succ S_R$, we first isolate an output variable r, this will give us terms $r \succ Q$ and $r \succ S$. There are two approaches which we now try in order to prove equivalence of $r \succ Q$ and $r \succ S$:

- **Algebraic equivalence**: With this approach we try to show that the expression $Q - S = 0$ using algebraic means. There are many cases were this approach will work, however it has been proved [27] that in general this problem is undecidable. Another approach involves substitution of r determined from the condition $r \succ S$ into $r \succ Q$, and subsequently proving their equivalence.
- **Value substitution**: With this approach we try to show that $Q - S = 0$ by substituting random values for each variable in the expression, then evaluating and checking to see if the valuation we get is zero. This is evidence that $Q - S = 0$, but is not conclusive, since we may have been unlucky in the case that the random values coincide with a zero of the expression.

3.4.2. Service Registry. The mathematical service descriptions are stored in a database comprising the following tables: service name, taxonomy, input, output, pre- and post-conditions, and omsymbol. For the matching of pre- and post-conditions, the tables omsymbol, precond and postcond are used. The other tables give additional details about a service once the matching is done, in order for the user to select the appropriate service from the returned list.

3.4.3. Matchmaker. For all services in the database, first the pre-conditions are read and for each the matching algorithm selected is applied—which returns a similarity value. For all similarity values of pre-conditions a match value is calculated and stored. The same procedure is then used for the post-conditions. For each service the match values for all pre- and post-conditions are calculated and stored together with the service details. The methods are those detailed in section 3.4.1 resulting in a match score in the interval $[0, 1]$.

4. Application Example. For the case study we only consider the four matching modes. The Factorisor service we shall look at is a service which finds all prime factors of an Integer. The Factorisor has the following post-condition:

```
<om:OMOBJ>
  <om:OMA>
    <om:OMS cd ='relation1' name ='eq'/>
    <om:OMV name ='n'/>
    <om:OMA>
      <om:OMS cd ='fns2' name ='apply_to_list'/>
      <om:OMS cd ='arith1' name ='times'/>
      <om:OMV name ='lst_fcts'/>
    </om:OMA>
  </om:OMA>
</om:OMOBJ>
```

where `n` is the number we wish to factorise and `lst_fcts` is the output list of factors.

As the structural and ontological modes compare the OpenMath structure of queries and services, and the algebraic equivalence and substitution modes perform mathematical reasoning, the case study needs to reflect this by providing two different types of queries.

For the structural and ontological mode let us assume that the user specifies the following query:

```
<om:OMOBJ>
  <om:OMA>
    <om:OMS cd ='fns2' name ='apply_to_list'/>
    <om:OMS cd ='arith1' name ='plus'/>
    <om:OMV name ='lst_fcts'/>
  </om:OMA>
</om:OMOBJ>
```

For the structural match, the query would be split into the following `OM` collection: `OMA`, `OMS`, `OMS`, `OMV` and `/OMA` in order to search the database with this given pattern. The match score of the post-condition results in a value of 0.27778 using the equations described earlier.

The syntax and ontology match works slightly different as it also considers the *values* of the `OM` symbols. In our example we have three `OM` symbol structures. There are two instances of `OMS` and one of `OMV`. First the query and the service description are compared syntactically. If there is no match, then the ontology match is called for the `OMS` structure. The value of the content dictionary (CD) and the value of the name are compared using the ontology of that particular CD. In this case the result is a match score of 0.22222. If the `OM` structure of the service description is exactly the same as the query then the structural match score is the same as for the syntax and ontology match.

The post-condition for the Factorisor service represents:

$$\mathtt{n} = \prod_{i=1}^{l} \mathtt{lst\_fcts}_i \qquad \text{where } \mathtt{l} = |\mathtt{lst\_fcts}| \tag{4.1}$$

Considering the algebraic equivalence and the value substitution, a user asking for a service with post-condition:[2]

$$\begin{aligned} \forall i|1 \leq i \leq |\texttt{lst\_fcts}| \Rightarrow n \bmod \texttt{lst\_fcts}_i = 0 \wedge \\ m \notin \{\texttt{lst\_fcts}_1, \cdots, \texttt{lst\_fcts}_l\} \Rightarrow n \bmod |m| \neq 0 \end{aligned} \tag{4.2}$$

should get a match to this Factorisor service.
To carry out the algebraic equivalence match we use a proof checker to show that:

- equation (4.1) $\Rightarrow$ equation (4.2): This is clear since the value of n (the RHS of equation 4.1) may be substituted into equation (4.2) and the resulting equality will be true for each value in `lst_fcts`.
- equation (4.2) $\Rightarrow$ equation (4.1): The first term in the conjunct holds by the definition of **mod**, whilst the second term says that there are no other numbers which divide n.

To compute the value substitution match we must gather evidence for the equivalense of expressions 4.1 and 4.2 by taking a number of random values and substituting these into the pre- and post-conditions, since we are checking the truth of relations, these will be trivially satisfied (or not). This will however only give us evidence for the equivalence of the conditions, as we may have chosen values for which the expressions just happened to be true. So this technique can only give a probability of correctness. In order to know the probability of correctness after a certain number of independent tests, it is necessary to know the size of the zero sets for the expressions which make up the pre- and post-conditions. Another limitation of this technique, is that (in its simplest form) it can only be applied to real valued expressions. We shall consider how the technique may be applied to the above problem:

- We first need to decide on the length of the list for our random example. A good basis would be to take $|\texttt{lst\_fcts}| = \lceil \log_2(\texttt{n}) \rceil$, this represents a bound on the number of factors in the input number.
- We then collect that number of random numbers, each of size bounded by $\sqrt{\texttt{n}}$.
- Then we calculate their product, from equation (4.1), this gives a new value for `n`.
- We may now check equation (4.2). We see that it is true for every value in `lst_fcts`.

If we try this for a few random selections, we obtain evidence for the equivalence of equations (4.1) and (4.2).

5. Measurements.

Of the four matching algorithms specified in section 3.4.1, our performance measurements are primarily focused on the "ontological match". This is because this mechanism provides the most computationally intensive match requirement. Structural match can already be undertaken with a variety of XML-based tools—that make use of XPath query, for instance. Similarly, algebraic equivalence also relies on the use of an ontological match to rewrite a mathematical expression prior to undertaking a match.

A set of measurements are described that: (1) evaluate the time it takes to load the MONET ontologies—briefly described in section 5.1, (2) the associated query response times and (3) the overall scalability of the ontological mode. Scalability analysis involved increasing the number of services hosted in the registry to 100,000.

5.1. The MONET Ontologies.

The ontology importation graph in Figure 5.1 essentially outlines the relationships between the various ontologies[3] (the Ontology Web Language (OWL) is used to describe each ontology) that were originally developed in the MONET project [31] for the purpose of demonstrating the basic end-to-end functionality from problem statement, through service discovery and invocation, to the delivery of results. It should be emphasized that with two notable exceptions—OpenMath and GAMS—these are not general-purpose ontologies but were engineered to meet a specific short-term need. Nevertheless, significant care went into their relative organization with a view to future developments. From the application's perspective, there is just the one ontology called MONET and that imports all the others, which is where we find the real content:

- **GAMS (Guide to Available Mathematical Software):** is described as "A cross-index and virtual repository of mathematical and statistical software components of use in computational science and engineering"[4] and provides a human browseable taxonomy of mostly numerical software, for example the extract given in Figure 5.2 is that used to classify numerical quadrature routines. As this extract makes clear, GAMS is represented as textual data and is intended for human consumption. Thus for program use an OWL analogue was generated making each node in the taxonomy into an OWL class, as shown in Figure 5.3.

[2]In this example, all factors are assumed prime (this could be given as another post-condition).
[3]`http://users.cs.cf.ac.uk/Simone.Ludwig/ontologies/monet/monet.owl`
[4]`http://gams.nist.gov/`

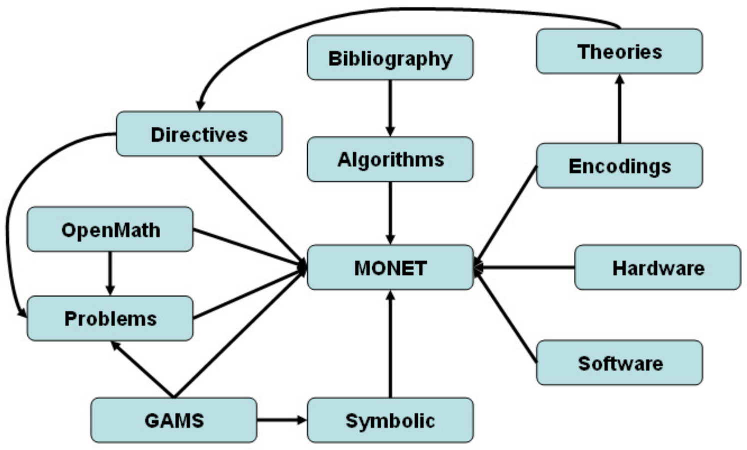

Fig. 5.1. *Relationship between MONET Ontologies*

- **Symbolic:** The GAMS taxonomy has a single category ("O") for symbolic computation that contains just four links to general-purpose computer algebra packages and one link to the ACM TOMS collected algorithms. Because a major focus of MONET was the combined application of numerical and symbolic techniques, this branch has undergone some extension to enable the description and advertisement of a range of symbolic mathematical services, such as group theory operations, indefinite integration, power series and algebraic geometry.
- **OpenMath:** This is not OpenMath[5] that we have used for mathematical semantic markup, but rather the interface to that ontology. Because OpenMath development started many years before OWL it does not (yet) use many of the current ontology technologies and is simply an XML-based encoding format for the representation of mathematical expressions and objects. Terms (referred to as "Constants") of the language have semantics attached to them and are called symbols (e.g., sin, integral, matrix, etc.), and groups of related symbols are collected in content dictionaries (CDs). What the OpenMath component of the MONET ontology defines is OWL classes for each of the symbols in the current OpenMath CDs.
- **Hardware:** This is used to describe either machine types or individual machines. The idea is that a user might request that a service run on a particular architecture (e.g. Sun Enterprise 10000), a general class of machine (e.g. shared memory), or a machine with a certain number of processors. Here is a fragment that describes a Sun shared memory machine:
```
<owl:Class rdf:ID="Enterprise10000">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#SharedMemory"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Sun"/>
  </rdfs:subClassOf>
</owl:Class>
```
- **Software:** allows a user to express a preference for a service that makes use of a particular programming language or software library, for example it represents a variety of programming languages (FORTRAN 77, FORTRAN 2000, C, etc.) and numerous packages (NAG library version 7 in C, Maple release 8, etc.).
- **Problems:** may be described in terms of inputs and outputs, pre-conditions and post-conditions, and make use of pre-defined XML schema [32]. Within this ontology, each problem is represented as a class, which can have properties indicating bibliography entries and their generalizations. The most interesting property is `openmath head` whose range is an object from the `OpenMathSymbol` class. This represents a particular symbol which can be used to construct an instance of the problem in question.

[5]`http://www.openmath.org`

```
H2.      Quadrature (numerical evaluation of definite integrals)
H2a.       One-dimensional integrals
H2a1.        Finite interval (general integrand)
H2a1a.         Integrand available via user-defined procedure
H2a1a1.          Automatic (user need only specify required accuracy)
H2a1a2.          Nonautomatic
```

Fig. 5.2. *An extract from the on-line GAMS taxonomy*

```
<owl:Class rdf:ID="GamsH2">
  <rdfs:comment>GAMS classification, Differentiation, integration,
                Quadrature (numerical evaluation of definite integrals)
  </rdfs:comment>
  <rdfs:label>Quadrature (numerical evaluation of definite integrals)</rdfs:label>
  <rdfs:subClassOf rdf:resource="#GamsH"/>
</owl:Class>
\dots
<owl:Class rdf:ID="GamsH2a">
  <rdfs:comment>GAMS classification, Differentiation, integration,
                Quadrature (numerical evaluation of definite integrals),
                One-dimensional integrals
  </rdfs:comment>
  <rdfs:label>One-dimensional integrals</rdfs:label>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#GamsH2"/>
  </rdfs:subClassOf>
</owl:Class>
```

Fig. 5.3. *An extract from the OWL representation of GAMS*

- **Algorithms:** there are two sub-classes in this ontology: (1) Algorithm: which describes well-known algorithms for mathematical computations, and (2) Complexity: which provides classes necessary for representing complexity information associated with an Algorithm.
- **Directives:** this ontology is a collection of classes which identify the task that is performed by the service as described in [32]—for example to decide, solve or prove a particular mathematical expression.
- **Theory:** this ontology collects classes that represent available formalized theories in digital libraries of mathematics.
- **Bibliography:** represents entries in well-known indices such as Zentralblatt MATH [33] and MathSciNet [34] and allows them to be associated with particular algorithms.
- **Encoding:** this ontology contains a (small) collection of classes which represent the formats used for encoding mathematical objects.
- **Monet:** As stated at the beginning of this section, MONET imports all the ontologies described above.

5.2. Methodology. The ontological mode of the matchmaker 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) [36] 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 [37], 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.

The implementation used for the ontological mode is the OWLJessKB version owljesskb20040223.jar [6]. The test environment included an Intel Pentium III processor 996MHz, 512MB RAM, and Windows XP Professional, running Java SDK 1.5.0 and Jess 6.1p8.

Measurements—Loading ontologies: These measurements include memory tests for OWLJessKB as heap size value were set for the loading of different ontology sizes. Additional measurements were carried out to evaluate the performance of loading the ontologies for the OWLJessKB implementation.

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.

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, scalability is evaluated as the change in query response behaviour as additional services were added to the registry.

5.3. Results.

5.3.1. Loading Ontology Performance. The previously described MONET ontologies (Figure 5.1) were chosen for the performance measurements. The MONET ontologies consist of 2031 classes, 78 slots and 10 facets. When increasing the ontology size, only the classes within the ontology were considered and expanded. Different ontology sizes were used having the number of classes as shown in Table 5.1.

Table 5.1
Ontology Sizes

Ontology size	No. of classes
1	2031
1.5	3046
2	4062
2.5	5077
3	6092
3.5	7107
4	8122

During preliminary measurement tests it was found that the OWLJessKB implementation needs the heap size in the Java Virtual Machine to be modified to load all ontology sizes. The first set of measurements were undertaken to investigate the necessary heap size required for the different ontologies. Figure 5.4 shows the memory heap size for varying ontology sizes. It shows a linear distribution starting from 109MB for ontology size 1 and ending at 847MB for ontology size 4. The regression line and the derived equation shown in the figure allow to calculate the memory heap size needed for a particular ontology size.

The maximum heap size was set to 1 GB, which was found to be a sufficient value for the measurements shown in Figure 5.4. The following set of measurements show the load time of the OWLJessKB implementation. Ten test runs were conducted for each ontology size.

5.3.2. Query Performance. The next set of measurements were carried out having four queries (simple assertion finding all instances of a class x, simple assertion asking whether instance x exists, assertion individual and assertion aggregate) to test the response times of varying ontology sizes for both target systems. The maximum heap size in the target system was set to 1 GB. For each query ten test runs were conducted and the average values were taken.

Queries for the OWLJessKB implementation need to be specified in the JESS notation, which is the following:

```
(defquery query-sub-class ‘‘Find all sub-classes"
  (triple
    (predicate ‘‘http://www.w3.org/2000/01/rdf-schema#
```

[6]http://edge.cs.drexel.edu/assemblies/software/owljesskb/

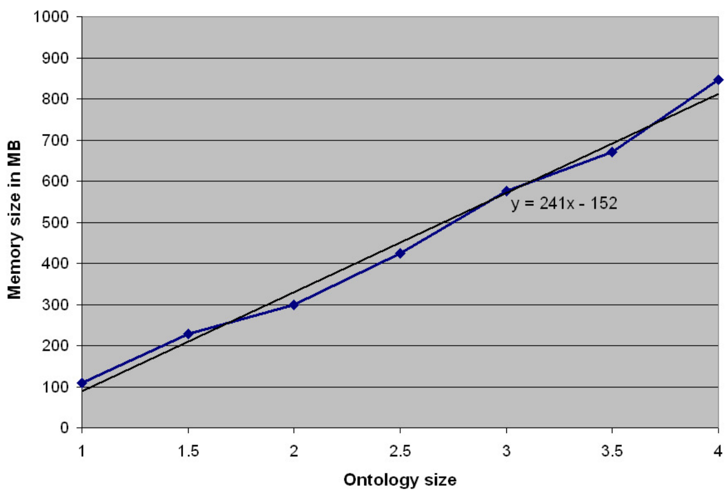

Fig. 5.4. *Memory heap size*

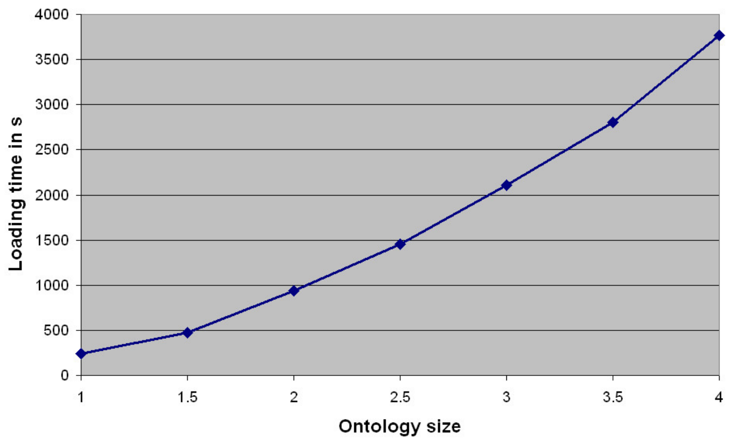

Fig. 5.5. *Load time*

```
              subClassOf")
    (subject ‘‘http://monet.nag.co.uk/owl#
            service_algorithm")
    (object ?y)))
```

A service stored in the registry is `nagopt`—fitting the GAMS taxonomy `G1a1a` [6] (a variant of unconstrained optimisation) and using OpenMath as its I/O format. In this case the description also indicates that the service uses NAG's implementation of the safeguarded quadratic-interpolation algorithm.

```
<service name="nagopt">
    <classification>
        <gams_class>GamsG1a1a</gams_class>
        <problem>constrained_minimisation</problem>
        <input_format>OpenMath</input_format>
        <output_format>OpenMath</output_format>
        <directive>find</directive>
    </classification>
    <implementation>
        <software>NAG_C_Library_7</software>
        <platform>PentiumSystem</platform>
```

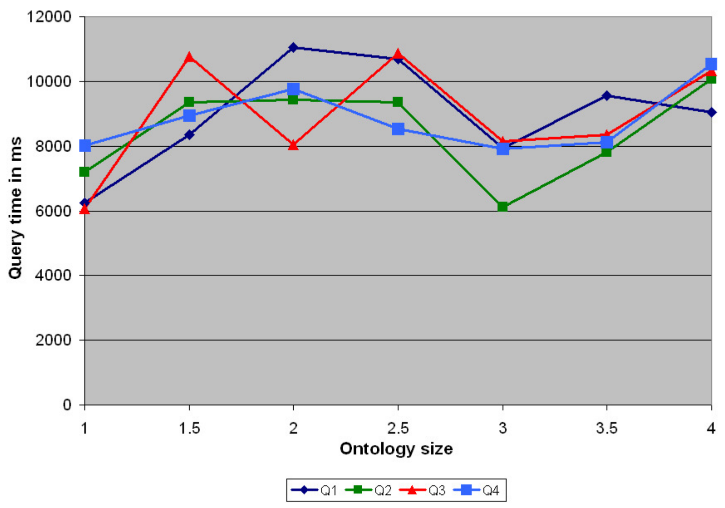

FIG. 5.6. *Query Performance*

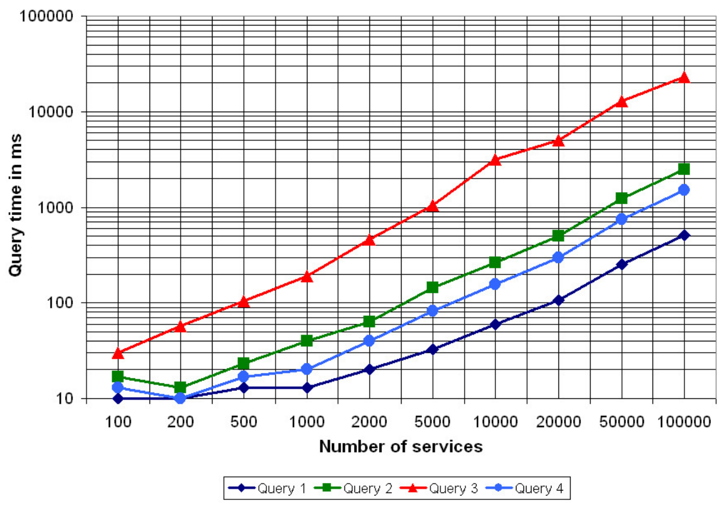

FIG. 5.7. *Scalability of populated services*

```
        <algorithm>Safeguarded_Quadratic-Interpolation</algorithm>
    </implementation>
</service>
```

In Figure 5.6 the response time of the four queries are shown. 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. Please note that Q1, Q4 times identical after ontology of size 2.

5.3.3. Scalability Performance. Measurements were taken populating the registry with an increasing number of services. Ontology size 1 was taken for this set of measurements, with a maximum heap size of 1 GB. The registry was populated with 100, 200, 500, 1,000, 2,000, 5,000, 10,000, 20,000, 50,000 and 100,000 services (therefore, Figure 5.7 use a logarithmic scale). In Figure 5.7 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.

6. Conclusion and Further Work. We have presented an approach to matchmaking in the context of mathematical semantics. The additional semantic information greatly assists in identifying suitable services in some cases, but also significantly complicates matters in others, due to their inherent richness. Consequently, we have put forward an extensible matchmaker architecture supporting the dynamic loading of plug-in matchers that may employ a variety of reasoning techniques, including theorem provers and computer algebra systems as well as information retrieval from textual documentation of mathematical routines (this last is under development at present). Although our set of application examples is as yet quite small, the results are promising and we foresee the outputs of the project being of widespread utility in both the e-Science and Grid communities, as well as more generally advancing semantic matchmaking technology. The performance measurements conducted showed the scalability of the system with the drawback of a relatively long ontology load time. Although the focus here is on matchmaking mathematical capabilities, the descriptive power, deriving from quantification and logic combined with the extensibility of OpenMath creates the possibility for an extremely powerful general purpose mechanism for the description of both tasks and capabilities. In part, this appears to overlap, but also to complement the descriptive capabilities of OWL and, in much the same way as it was applied in MONET, we expect to utilise OWL reasoners as plug-in matchers in the architecture we have set out.

REFERENCES

[1] S. A. Ludwig, O. F. Rana, J. Padget and W. Naylor, *Matchmaking Framework for Mathematical Web Services,* Journal of Grid Computing, Springer Verlag, 2006.

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

[3] K. Sycara and S. Widoff and M. Klusch and J. Lu, *Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace,* Journal of Autonomous Agents and Multi Agent Systems, Kluwer, 2002.

[4] V. R. Benjamins and B. Wielinga and J. Wielemaker and D. Fensel, *Towards Brokering Problem-Solving Knowledge on the Internet,* Ed. Dieter Fensel and Rudi Studer, Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW-99), LNAI1621, Springer, 1999.

[5] M. Gomez and E. Plaza, *Extended matchmaking to maximize capability reuse,* Proceedings of The Third International Joint Conference on Autonomous Agents and Multi Agent Systems, ACM Press, 2004.

[6] MONET Consortium, MONET Home Page, Available from \tthttp://monet.nag.co.uk 2002.

[7] D. Kuokka and L. Harada, *Matchmaking for information integration,* Journal of Intelligent Information Systems, Feb 1996.

[8] O. Caprotti and M. Dewar and J. Davenport and J. Padget, *Mathematics on the (Semantic) Net,* Proceedings of the European Symposium on the Semantic Web, Springer Verlag, 2004.

[9] The GENSS Project, GENSS Home Page, Available from `http://genss.cs.bath.ac.uk` 2004.

[10] OpenMath Society, OpenMath website, Available from `http://www.openmath.org`, 2002.

[11] W3C MathML, *Mathematical Markup Language (MathML) Version 2.0,* W3C, Available from `http://www.w3.org/TR/MathML2/`, 2003.

[12] W. Naylor and S. Watt, *Meta-stylesheets for the conversion of mathematical documents into multiple forms,* Annals of Mathematics and Artificial Intelligence Journal, vol. 38, no. 1–3, May, 2003.

[13] S. Buswell and O. Caprotti and M. Dewar, *Mathematical Service Description Language,* Available from the MONET website: `http://monet.nag.co.uk/cocoon/monet/publicdocs/monet-msdl-final.pdf`, 2003.

[14] A. Ankolekar, M. Burstein, J.R. Hobbs, O. Lassila, D.L Martin, S.A. McIlraithe, S. Narayanan, M. Paolucci, T. R. Payne, K. Sycara, H. Zeng, *DAML-S: Semantic Markup for Web Services,* Proceedings of 1st International Semantic Web Conference (ISWC 02), 2002.

[15] W3C Coalition, *OWL-S: Semantic Markup for Web Services,* In Technical White paper (OWL-S version 1.0), 2003.

[16] V. R. Benjamins, E. Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal and S. Decker, *An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web,* Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98), Banff, Canada, 1998.

[17] R. J. Brachman and J. G. Schmolze, *An overview of the KL-ONE knowledge representation system,* Cognitive Science, 9(2): 171–216, 1985.

[18] The OpenMath Society, The OpenMath Standard, The OpenMath Society, Available from `http://www.openmath.org/cocoon/openmath/standard/om20/index.html`, 2004.

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

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

[21] M. Genesereth and R. Fikes, *Knowledge interchange format,* version 3.0 reference manual, Computer Science Department, Stanford University, Available from `http://www-ksl.stanford.edu/knowledge-sharing/papers/kif.ps`, 1992.

[22] G. Salton, *Automatic Text Processing,* Addison-Wesley, 1989.

[23] D. Veit, *Matchmaking in Electronic Markets,* An Agent-Based Approach towards Matchmaking in Electronic Negotiations, Springer, LNCS2882, 2003.

[24] A. Moormann Zaremski and J. M. Wing, *Specification Matching of Software Components,* ACM Transactions on Software Engineering and Methodology, 1997.

[25] J. F. Sowa, *Ontology, Metadata, and Semiotics, Conceptual Structures: Logical, Linguistic, and Computational Issues,* Lecture Notes in AI #1867, Springer-Verlag, 2000.

[26] J. Kopena, *OWLJessKB,* Available at `http://edge.cs.drexel.edu/assemblies/software/owljesskb/`, 2004.

[27] D. Richardson, *Some Unsolvable Problems Involving Elementary Functions of a Real Variable,* Journal of Computational Logic, 1968.

[28] W3C WSDL, Web Services Description Language (WSDL) 1.1, Available from `http://www.w3.org/TR/wsdl,` 2004.

[29] L. Verlet, *Computer Experiments on Classical Fluids I. Thermodynamical Properties of Lennard-Jones Molecules,* Phys. Rev., Vol. 159, pp. 98–103, 1967.

[30] Maozhen Li, O.F.Rana, David W. Walker, *Wrapping MPI-Based Legacy Codes as Java/CORBA Components,* Future Generation Computer System(FGCS): The Int. Journal of Grid Computing: Theory, Methods and Applications, vol. 18, Issue 2, pages 213–223, October 2001.

[31] O. Caprotti, M. Dewar and D. Turi, *Mathematical service matching using Description Logic and OWL,* In Proceedings of 3rd Int'l Conference on Mathematical Knowledge Management (MKM'04), Vol:3119, pages 144–151. Springer-Verlag, 2004.

[32] O. Caprotti, D. Carlisle, A. Cohen and M. Dewar, *Problem Ontology: final version,* The MONET Consortium Technical Report Deliverable D11. Available from: `http://monet.nag.co.uk`

[33] Zentralblatt MATH. Available from: `http://www.emis.de/ZMATH/`

[34] American Mathematical Society, *MathSciNet: Mathematical Reviews on the Web.* Available from: `http://www.ams.org/mathscinet`.

[35] 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, Poznań, Poland, September 2005.

[36] Ernest J. Friedman-Hill, *Java Expert Systems Shell.* Available from: `http://herzberg.ca.sandia.gov/jess/docs/61/index.html`

[37] C. Forgy, *Rete: A fast algorithm for the many pattern/many object pattern match problem.* Journal of Artificial Intelligence, 19: 17–37, 1982.

[38] 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.

Edited by: Przemysław Stpiczyński.
Received: April 6, 2006.
Accepted: May 28, 2006.