Journal of Grid Computing (2006) 4: 33–48
DOI: 10.1007/s10723-005-9019-z

Matchmaking Framework for Mathematical Web Services

Simone A. Ludwig[1,⋆], Omer F. Rana[1], Julian Padget[2] and William Naylor[2]
[1]*School of Computer Science, Cardiff University, 5 The Parade, Roath, Cardiff, CF24 3AA, UK*
E-mail: Simone.Ludwig@cs.cardiff.ac.uk
[2]*Department of Computer Science, University of Bath, Bath, UK*

Received 9 August 2005; accepted in revised form 13 December 2005

Key words: match plug-ins, match score, matchmaking, mathematical web services

Abstract

Service discovery and matchmaking in a distributed environment has been an active research issue for some time now. 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 and for which we use OpenMath descriptions of pre- and post-conditions. We describe a matchmaking architecture supporting the use of match plug-ins and describe five kinds of plug-in that we have developed to date: (i) A basic structural match, (ii) a syntax and ontology match, (iii) a value substitution match, (iv) an algebraic equivalence match and (v) a decomposition match. The matchmaker uses the individual match scores from the plug-ins to compute a ranking by applicability of the services. We consider the effect of pre- and post-conditions of mathematical service descriptions on matching, and how and why to reduce queries into Disjunctive Normal Form (DNF) before matching. A case study demonstrates in detail how the matching process works for all four algorithms.

1. Introduction

The amount of machine-oriented data on the Web is increasing rapidly as semantic Web technologies achieve greater up-take. At the same time, the deployment of agent/Web Services is increasing and together create a problem for software agents that is the analog of the human user searching for the right HTML page. Humans typically use Google, but they can filter out the irrelevant and spot the useful, so while UDDI (the Web Services registry) with keyword searching essentially offers something similar, it is a long way from being very helpful. Consequently, there has been much research on intelligent brokerage, such as Infosleuth [18], LARKS [22], 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.
- *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

⋆ Corresponding author.

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 [11] by the following condition:

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

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 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 [7, 15] and GENSS [24] 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 [25] 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. Our key motivation in using mathematical services is that the provision of support for discovering such services may be more useful than the capability to provide services within a particular domain (such as BioInformatics, for instance). Numerical services are also 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 semantic matching for mathematical services in which we put forward a polyalgorithmic approach to answering the implication question posed above. The rest of the paper is laid out as follows: 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 a case study which examines integer factorisation and in Section 5 we summarise and conclude this paper.

1.1. *Usage Scenarios*

Within existing Grid applications, there are a number of possible modes of use. For instance, a predominant use within the high performance computing community involves a user submitting a single large application to a remote compute service, and then getting back the results on completion. However, a significant benefit of the Grid computing vision is the capability to combine results from multiple services (often not co-located), aggregate the results and send these back to the user. Generally, such an approach involves the use of a workflow enactment engine to coordinate the execution of services, and an overall workflow system that allows the configuration of the engine, a way to define the workflow graph, and subsequently to present results in some meaningful way to the user. The matchmaking approach presented here would therefore be useful in two types of scenarios:

- Prototyping: In this instance, a user may be interested in evaluating different mathematical techniques in the context of a known problem. The user may not have access to a suitable mathematical algorithm to solve the problem, or may be interested in identifying alternative techniques that could provide a solution. In such a scenario, a search for a suitable service may be undertaken in a localized manner, involving NAG libraries that a user has direct access to – for instance, or may involve connecting to multiple known registry services owned by other users. The matchmaking approach presented would be most relevant in such a setting.
- Workflow composition: A workflow system generally involves a set of pre-defined services (made available when a workflow system is initialized) that can be connected together to create an application. Generally, once services have been combined into a graph, execution of

which is then via an enactment engine. Recent efforts in workflow management however also involve dynamic updating of the list of services within a workflow system [23]. Such discovery currently involves periodically connecting to a known registry service, and updating the list of available services within the workflow system.

The matchmaking approach presented here can also add value within such a workflow system, by allowing a user to search for services on-demand. A matchmaker is provided as just another service in the workflow system, but one which can be used to discover new services. Once a suitable service has been identified, a reference to this service can be added to the workflow system – and subsequent enactment can be carried out as before.

Currently, no data is made available on performance information associated with a matched service in our system. The current focus is therefore primarily on finding suitable services based on their functionality. Provided suitable monitoring tools are available (such as Ganglia [1]) within the environment hosting a service, our existing service description can be extended to also support such non-functional properties of a service [12].

2. Description of Mathematical Services

2.1. *OpenMath and MathML*

In order to describe mathematics and to allow mathematical objects to be exchanged between computer programs, stored in databases, or published on the worldwide web an emerging standard called OpenMath [25] has been introduced. 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 [25], 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 whilst the variables and integers speak for themselves. 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 ([29] Section 5.1) and [17] 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 ([29] Section 5.3).

There are various ways in which OpenMath can help in matchmaking:

- 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

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

Mathematical Service Description Language (MSDL) [6].

MSDL is an extension of WSDL that was developed as part of the MONET project [15], incorporating more information about a service, in particular pre- and post-conditions, taxonomic references etc. An example MSDL document describing a factorisation service for square-free integers is shown in Figure 1.[2] OpenMath is used to describe the service semantics, but other encodings are possible.

3. Mathematical Matchmaking

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 [2] and OWL-S [8]? 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 [18], RETSINA [22] and IBROW [3], 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 [25] recently (May 2004) celebrated its 10th anniversary, which indicates how early an application of XML it was, when many of the features we know today did not exist and long before RDF came about – although long after the birth of KLONE [5]. Nevertheless, OpenMath stands as probably the most developed ontology of mathematics, because in contrast to MATHML-C [29] 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 [26] replaces DTDs with schemas, seeks 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 shall continue to use 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 [14] 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 [9]. Content languages of SHADE are a subset of KIF (Knowledge Interchange Format) [10] 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.

COINS (COmmon INterest Seeker) [14] is a matchmaker which operates over free text. The

[2] Throughout the paper, the prefix `monet` is used to denote the namespace: `http://monet.nag.co.uk/monet/ns`.

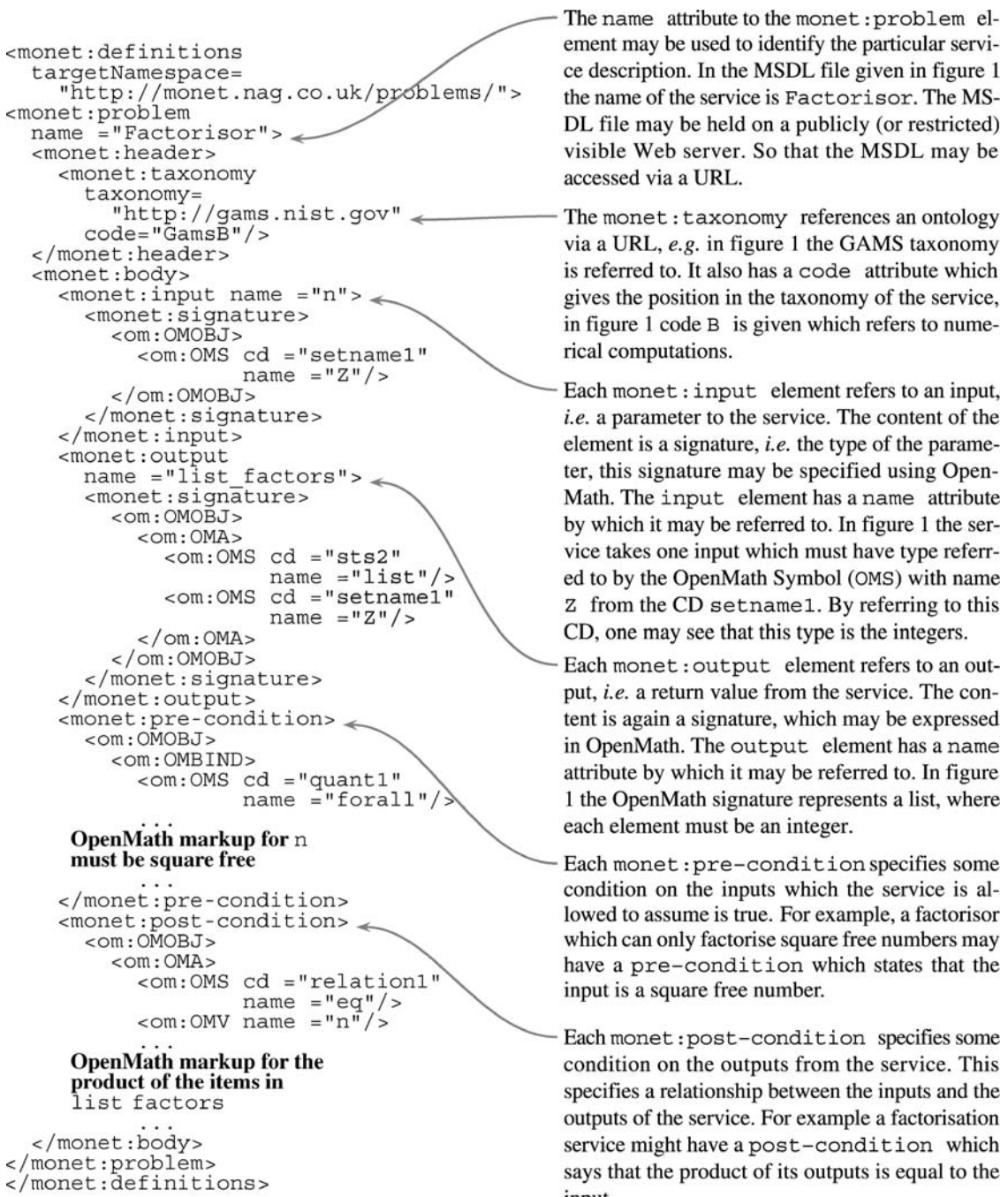

Figure 1. MSDL description of a factorisation service.

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) [20] 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) [22] 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 [18] 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 [28] (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.

MathBroker is a project at RISC-Linz with some elements in common with those described here, including providing semantic descriptions of mathematical services. It too uses MSDL, however it seems that most of the matchmaking is achieved through traversing taxonomies, while actual understanding of 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 [31]. 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. Furthermore, we believe that 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. 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 2 and comprises the following parts:

- Client interface: Which may be employed by the user to specify their service request.
- Matchmaker: Which contains a reasoning engine and the matching module.
- UDDI registry: Where the matching algorithm Web services are registered.
- Matching algorithm web services: Where the logic of the matching is defined.
- Mathematical ontologies: Such as OpenMath CDs, GAMS etc.
- Registry service: Where the mathematical service descriptions are stored.
- Mathematical web services: Available on third party sites, accessible over the Web.

The sequence diagram in Figure 3 shows the interactions between these components of a service request. The interactions of a search request are as follows:

- The user contacts the matchmaker, then

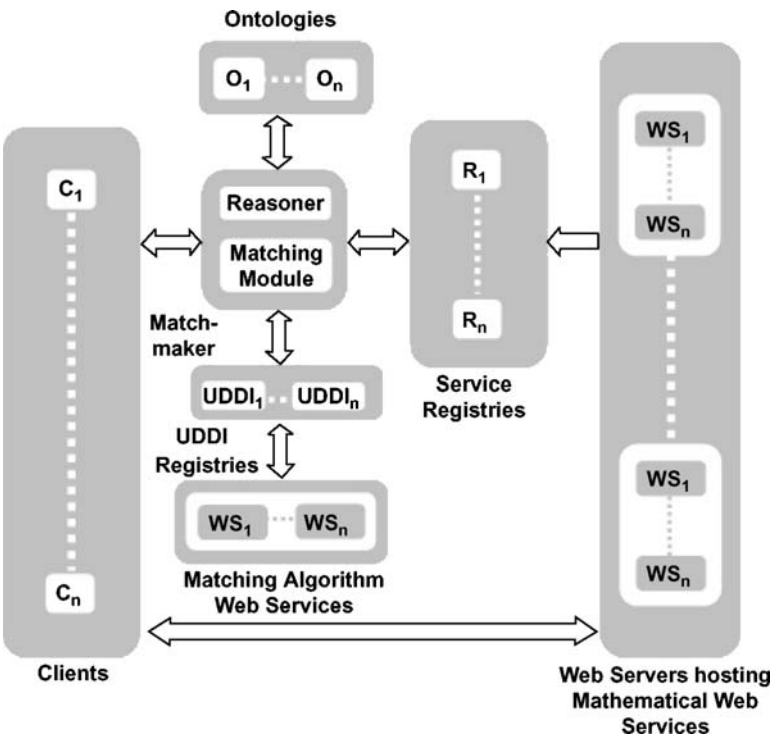

Figure 2. Matchmaking architecture.

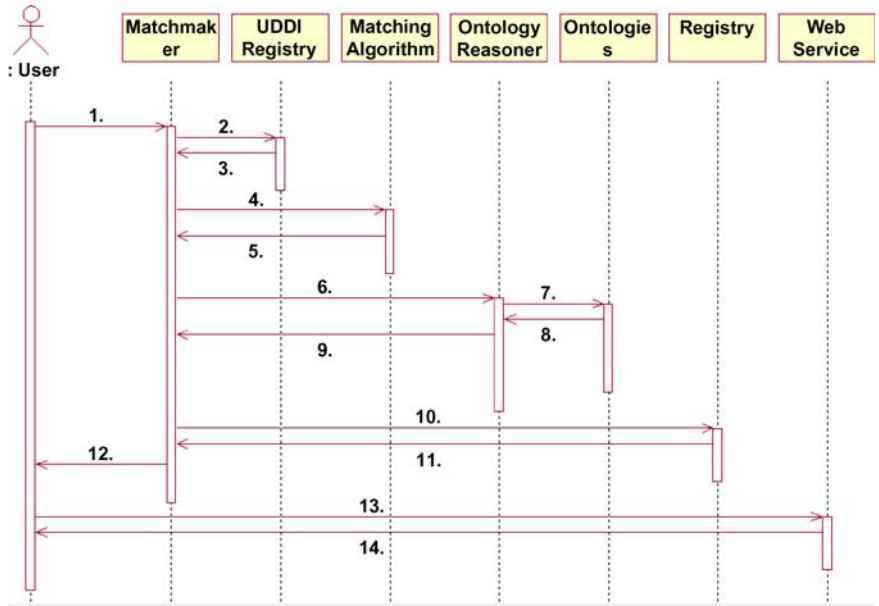

Figure 3. Sequence diagram – service request.

- the matchmaker loads the matching algorithms specified by the user via a lookup in the UDDI registry; 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.

The parameters stored in the registry (a database) are service name, URL, taxonomy, 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. *Client*

The Client application[3] (shown in Figure 4) allows the user to specify the service request via entry fields for pre- and post-conditions. The matchmaker returns the matches in the table at the bottom of the GUI listing the matched services ranked by similarity. Subsequently the user can invoke the service by clicking on the URL.

3.4.2. *Matching Algorithms*

Currently five matching algorithms have been implemented within the matchmaker:

- Structural match
- Syntax and ontological match
- Algebraic equivalence match
- Value substitution match
- Decomposition 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.

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

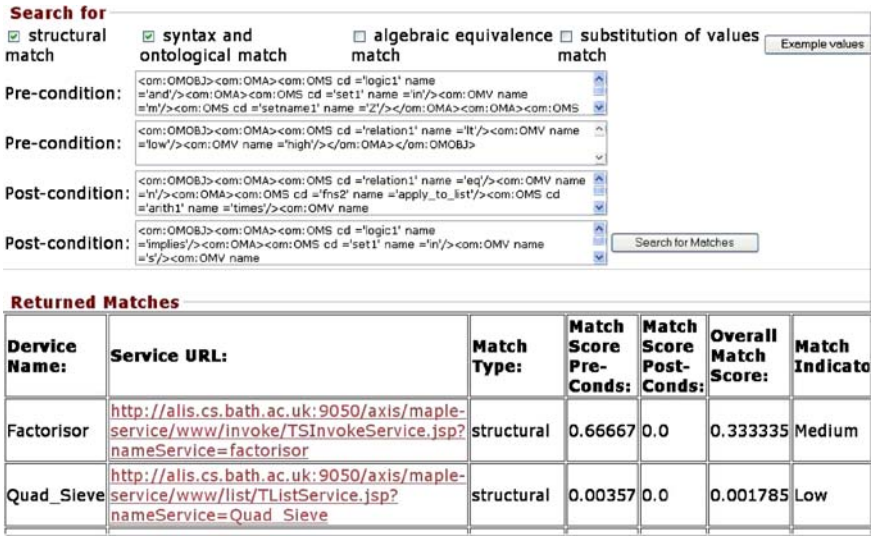

Figure 4. Matchmaker client application.

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 [21] 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 5. OWLJessKB [13] 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'/>.
```

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 degree of separation of the concepts or in other words the distance between the two concepts.

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

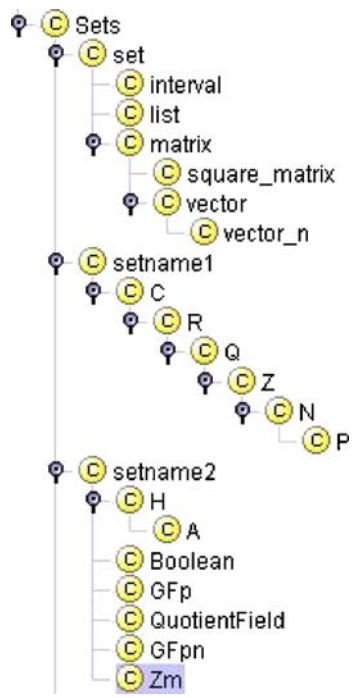

Figure 5. Set ontology fragment.

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"/>
    <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., $\forall x \exists y \text{s.t.} \cdots$
- L is a block of logical connectives, e.g., $\wedge, \vee, \Rightarrow, \cdots$
- R is a block of relations, e.g., $=, \leq, \geq, \neq, \cdots$

Thus the objective of normalization is to put the task and capability pre- and post-conditions in a form for effective comparison. While there is no right representation, as long as we normalize task and capability using the same techniques, they can be compared and a similarity value computed. Each of the above parts of the expression is handled as follows:

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.

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. We choose to transform the query logical block into DNF (disjunctive normal form), that is a form which only contains a disjunction of terms and the negation of terms: $\bigvee_i QT_i \bigvee_j \overline{QT_j}$, which means the conditions are represented as a set of alternatives, making the comparison relatively easy, not least because superfluous conditions do not then affect the eligibility test. The capability description is transformed likewise ($\bigvee_i ST_i \bigvee_j \overline{ST_j}$), and consequently, we can define what matching such terms means. There are four possible combinations:

- A term in QT_i matches a term in ST_i. This should count as a positive match.
- A term in QT_i matches a term in $\overline{ST_j}$. This should count as a negative match.
- A term in $\overline{QT_j}$ matches a term in ST_i. This should count as a negative match.
- A term in $\overline{QT_j}$ matches a term in $\overline{ST_j}$. This should count as a positive match.

A more detailed discussion of the semantics of matching mathematical descriptions appears in [16].

The relations block: We now have a disjunction of terms which we are matching against a set of conjunction of terms. 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 [19] 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.

The **decomposition match** attempts to discover an equivalent mathematical expression in case an exact service match cannot be found. Essentially, this involves dividing the query into sub-queries, and trying to find a match for each decomposed sub-query. The decomposition is supported by applying a set of rules that try to match each service description. The rules are applied recursively to decompose a mathematical expression into its simplest form. If we take the previous example, we see that evaluating the expression $x^2 - y^2$ at specific values for x and y could be decomposed into evaluations of the simpler expressions $e_1 = x - y$, $e_2 = x + y$ and $r = e_1 * e_2$.

3.4.3. *UDDI Registry*

The Web Services Description Language (WSDL) [30] is an XML-based language used to describe a Web service. This description allows an application to dynamically determine a Web service's capabilities, which are for example, the operations it provides, their parameters, return values, etc. A UDDI [27] repository is a searchable directory of Web services that Web service requestors can use to search for Web services and obtain their WSDL documents. UDDI consists of three components: 'White pages' to hold basic contact information and identifiers for a company, 'yellow pages' to enable companies to be listed based on their industry categories (using standard taxonomies), and 'green pages' to record interface details of how a Web service is to be invoked. Providers may register services with one or more registries (using the same identifier) – and may be discovered by search distributed over one or more registries.

Figure 6 shows the registered matching algorithm Web services which are Structural Matcher, Ontology Matcher, Substitution Matcher, Algebraic Matcher and Decomposition Matcher. The business where they are registered is `GENSS Project`. The industry classification chosen for our matching algorithm services is the NAICS (North American Industry Classification System) standard which is encoded in the `tModelKey`. All registered Web Services have an access point which is the WSDL document.

The following paragraph describes the process of registering a new matching algorithm in the Cardiff UDDI. It is necessary to connect to UDDI[4] and register the matching algorithm Web Service with the business key [5] shown in the binding template in Figure 7. The `authtoken` provides a way to authenticate a particular session with the Web server. The business key is obtained when registering a new Business in the UDDI – and generally associated with particular categories that a given Business may be assigned to. However, in order to make the matching algorithm work with the matchmaker, the matching algorithm needs to follow a defined interface description, whereby the passed argument is a `DataHandler` object containing a vector of pre-conditions and a vector of post-conditions, and the return argument is a `DataHandler` object consisting of a vector of `ServiceMatchDetails` objects. In our example using the 'white' pages search word `GENSS` would result in querying the business information

[4] http://uddi.wesc.ac.uk:8334/juddi/.

[5] businessKey="226BB8A0-6BC3-11D9-B8A0-865238849EB8".

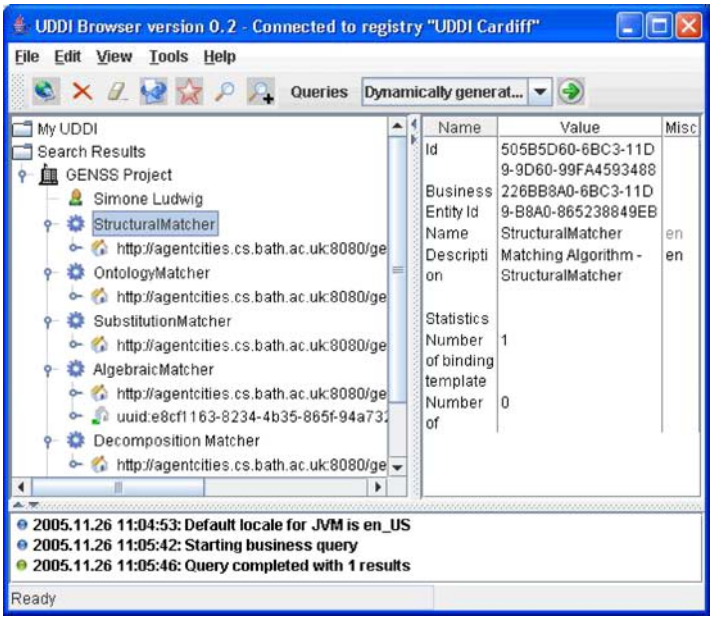

Figure 6. Matching algorithm web services registered in UDDI registry.

encoded in the `businessKey` returning the five matching algorithm services. This allows a dynamic loading of matchmaking algorithms in order to allow the matchmaker to be driven in the different matching modes. Using the 'yellow' pages search with the keyword `OntologyMatcher` would return the WSDL document of the Web Service accordingly. As illustrated in Figure 7, the `URL` points to the location of the service interface – accessed by a Web Service client to invoke the service.

3.4.4. *Service Registry*

The mathematical service descriptions are stored in a database comprising the following tables: service, taxonomy, input, output, precond and postcond, 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.

3.4.5. *Matchmaker*

Algorithm 1 *Matchmaking*

PrCQ: Pre-conditions of query
PoCQ: Post-conditions of query
PrCS: Pre-conditions of service
PoCS: Post-conditions of service
SVPrC: Similarity values of Pre-conditions
SVPoC: Similarity values of Post-conditions
MVPrC: Match values of Pre-conditions
MVPoC: Match values of Post-conditions
MVO: Overall match score of service
SD: Service details
MD: Match details of service

PrCQ ← *read_In_PreConds*()
PoCQ ← *read_In_PostConds*()
connect_To_DB()
for *all_service_In_DB* **do**
  PrC ← *read_PreConds_From_DB*()
  for {*PrCS*} **do**
    SVPrC ← *select_Match_Algo*()
  end for
  MVPrC ← *calculate_Match_Value*()
  PoCS ← *read_PostConds_From_DB*()
  for *PoCS* **do**
    SVPoC ← *select_Match_Algo*()
  end for
  MVPoC ← *calculate_Match_Value*()
  MVO ← *calculate_Match_Score*()
  SD ← *retrieve_Service_Details*()
  MD ← *store_Match_Details*()
end for
disconnect_From_DB()
return *MD*

```
<save_service generic="2.0xmlns="urn:uddi-orh:api_v2>
  <authInfo>authToken:5BEAB560-6CB1-11D9-B560-BC7A86A797A5</authInfo>
  <businessService businessKey="226BB8A0-6BC3-11D9-B8A0-865238849EB8"
    serviceKey="90B25DC0-6CB1-11D9-9DC0-90000B8354DD">
    <name xml:lang="en">Algebraic Matcher</name>
    <description xml:lang="en">Matching Algorithm -
                  AlgebraicMatcher</description>
    <bindingTemplates>
      <bindingTemplate
        bindingKey="uuid:e8cf1163-8234-4b35-865f-94a7322e40c3">
        <accessPoint URLType="http">
          http://agentcities.cs.bath.ac.uk:8080/genss_axis/
          services/AlgebraicMatcher</accessPoint>
        <tModelInstanceDetails>
          <tModelInstanceInfo
            tModelKey="uuid:6e090afa-33e5-36eb81b71ca18373f457">
            <instanceDetails>
              <overviewDoc>
                <overviewURL>
                  http://agentcities.cs.bath.ac.uk:8080/genss_axis/
                  services/AlgebraicMatcher?wsdl</overviewURL>
              </overviewDoc>
            </instanceDetails>
          </tModelInstanceInfo>
        </tModelInstanceDetails>
      </bindingTemplate>
    </bindingTemplates>
  </businessService>
</save_service>
```

Figure 7. UDDI binding template.

The matchmaking algorithm is specified in Algorithm 1. The pre- and post-conditions are read in first. Then a connection to the database is made. 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 overall consideration within the matchmaking approach is to get a match score returned which should be between 0 and 1, where 0 represents no match and 1 represents an exact match (1). Looking at the pre- and post-conditions separately, it is first of all necessary to determine the ratio of the number of pre-conditions given in the query in relation to the number given by the actual service where some or all pre- or post-conditions match. To make sure that this ratio does not exceed 1, a normalisation is performed with the inverse of the sum of both values. This is multiplied by the sum of the similarity values for each match of a pre-condition divided by the number of actual matches in order to keep the overall score value between 0 and 1 in Equation (2). The same is done with the post-conditions (3). The importance of the pre- or post-conditions is reflected in the weight values. The match scores may be calculated using the following equations:

$$M_O = \frac{M_A + M_B}{2} \tag{1}$$

$$M_A = \frac{w_a}{|A_Q| + |A_S|} * \frac{|A_Q|}{|A_S|} * \frac{\sum_{i=1}^{|A|}(SV_A(i))}{|A|}$$
$$\text{where } w_a + w_b = 1 \tag{2}$$

$$M_B = \frac{w_b}{|B_Q| + |B_S|} * \frac{|B_Q|}{|B_S|} * \frac{\sum_{j=1}^{|B|}(SV_B(j))}{|B|}$$
$$\text{where } w_a + w_b = 1 \tag{3}$$

In the above, M_O, M_A, M_B are the overall, the pre-condition and the post-condition match scores respectively. $|\{c\}|$ denotes the number of conditions in $\{c\}$, A_Q and A_S are pre-conditions, B_Q and B_S are post-conditions, the subscripts Q and S refer to the queries and services respectively. A, B are a set of matched pre-conditions, post-conditions respectively and $SV_A(i)$, $SV_B(j)$ are the similarity values for the ith matched pre-condition and the jth matched post-condition respectively.

4. Case Study

For the case study we only consider the first 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 (using the equations as stated above):

$$M_A = \frac{0.5}{1+1} \times \frac{1}{1} \times \frac{\frac{5}{9}}{1} = 0.13889$$

whereby $\frac{5}{9}$ is the number of items in the collection divided by the number of items in the pre-condition of the service, and $M_B = 0$, thus $M_O = 0.06944$.

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:

$$M_A = \frac{0.5}{1+1} \times \frac{1}{1} \times \frac{\frac{4.1}{9}}{1} = 0.11389$$

whereby fraction 4.1 is the similarity value (four items in the collection match and the similarity value measured via the ontology is 0.1). $M_B = 0$ and therefore M_O results in a value of 0.05695.

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:

$$\mathtt{n} = \prod_{i=1}^{l} \mathtt{lst\_fcts}_i \quad \text{where} \quad \mathtt{l} = |\mathtt{lst\_fcts}| \tag{4}$$

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

$$\forall i | 1 \leq i \leq |\mathtt{lst\_fcts}| \Rightarrow n \mod \mathtt{lst\_fcts}_i = 0 \tag{5}$$

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) ⇒ Equation (5): This is clear since the value of n may be substituted into Equation (4) and the resulting equality will be true for each value in `lst_fcts`.

— Equation (5) ⇒ Equation (4): A slightly stronger version of Equation (5) which says that there are no other numbers which divide *n*.

To compute the value substitution match we must gather evidence for the truth of Equations (4) and (5) by considering a number of random examples, we proceed as follows:

— 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), this gives a new value for n.
— We may now check Equation (5). 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) and (5).

5. 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 test cases 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. 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. Furthermore, we are currently running experiments for measuring the scalability of this framework in comparison with another matchmaking framework called the instanceStore. Preliminary findings suggest that the scalability of this framework is much higher than the instanceStore when searching for the right service. However, the initialisation (loading of the ontology) in turn is much slower.

Acknowledgements

The work reported here is partially supported by the Engineering and Physical Sciences Research Council under the Semantic Grids call of the e-Science program (grant reference GR/S44723/01).

References

1. "Ganglia Monitoring System", http://ganglia.sourceforge.net/.
2. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraithe, S. Narayanan, M. Paolucci, T. Payne, K. Sycara and H. Zeng, "DAML-S: Semantic markup for web services", in *Proc. 1st Int'l Semantic Web Conf. (ISWC 02)*, 2002.
3. 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-wideweb", in *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'98),* Banff, Canada, pp. 18–23, April 1998.
4. V.R. Benjamins, B. Wielinga, J. Wielemaker and D. Fensel, "Towards brokering problem-solving knowledge on the internet", in D. Fensel and R. Studer (eds.), *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management* (EKAW-99), Vol. 1621 of *LNAI*, Springer, Berlin Heidelberg New York pp. 33–48, 1999.
5. R. Brachman and J. Schmolze, "An overview of the KL-ONE knowledge representation system", 1985.
6. S. Buswell, O. Caprotti and M. Dewar, "Mathematical Service Description Language", Technical Report, 2003. Available from the MONET website: http://monet.nag.co.uk/cocoon/monet/publicdocs/monet-msdl-nal.pdf.
7. O. Caprotti, M. Dewar, J. Davenport and J. Padget, "Mathematics on the (Semantic) Net", in C. Bussler, J.

Davies, D. Fensel, and R. Studer (eds.), *Proceedings of the European Symposium on the Semantic Web*, Vol. 3053 of *LNCS*, Springer, pp. 213–224, 2004.
8. Coalition, "OWL-S: Semantic markup for web services", 2003.
9. T. Finin, R. Fritzson, D. McKay and R. McEntire, "KQML as an agent communication language", in *Proceedings of 3rd International Conference on Information and Knowledge Management*, pp. 456–463, 1994.
10. 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.
11. M. Gomez and E. Plaza, "Extended matchmaking to maximize capability reuse", in N. R. Jennings, C. Sierra, L. Sonnenberg and M. Tambe (eds.), *Proceedings of The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, Vol. 1., ACM, pp. 144–151, 2004.
12. L. Huang, D. Walker, Y. Huang and O. Rana, "Dynamic web service selection for workflow optimisation", *Proceedings of UK eScience All-Hands Meeting, Nottingham*, 2005.
13. J. Kopena, "OWLJessKB", 2004. http://edge.cs.drexel.edu/assemblies/software/ owljesskb/.
14. D. Kuokka and L. Harada, "Integrating information via matchmaking", *Intelligent Information Systems*, Vol. 6, No. 2–3, pp. 261–279, 1996.
15. MONET Consortium, "MONET Home Page", 2002, Available from http://monet.nag.co.uk.
16. W. Naylor and J. Padget, "Semantic matching for mathematical services", *Lecture Notes in Computer Science*, Vol. 3863, pp. 174–189, 2006.
17. W. Naylor and S. Watt, "Meta-stylesheets for the conversion of mathematical documents into multiple forms", *Annals of Mathematics and Artificial Intelligence*, Vol. 38, No. 1–3, pp. 3–25, 2003.
18. M. Nodine, W. Bohrer, and A. Ngu, "Semantic brokering over dynamic heterogenous data sources in InfoSleuth", in *Proceedings of the 15th International Conference on Data Engineering*, pp. 358–365, 1999.
19. D. Richardson, "Some unsolvable problems involving elementary functions of a real variable", *Journal of Computational Logic*, Vol. 33, pp. 514–520, 1968.
20. G. Salton, *Automatic Text Processing*, Addison-Wesley, 1989.
21. J. Sowa, "Ontology, Metadata, and Semiotics, Conceptual Structures: Logical, Linguistic, and Computational Issues", Lecture Notes in AI #1867, Springer, Berlin Heidelberg New York pp. 55–81, 2000.
22. K. Sycara, S. Widoff, M. Klusch and J. Lu, "Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace", *Journal of Autonomous Agents and Multi Agent Systems*, Vol. 5, No. 2, pp. 173–203, 2002.
23. I. Taylor, M. Shields, I. Wang and O. Rana, "Triana applications within Grid computing and peer to peer environments", *Journal of Grid Computing*, Vol. 1, pp. 199–217, 2003.
24. The GENSS Project, "GENSS Home Page", 2004. Available from http://genss.cs.bath.ac.uk.
25. The OpenMath Society, "The OpenMath Standard", 2002. Available from http://www.openmath.org/standard/om11/omstd11.xml.
26. The OpenMath Society, "The OpenMath Standard", 2004. Available from http://www.openmath.org/cocoon/openmath/standard/om20/index.html.
27. UDDI, "UDDI Technical White Paper", 2003, Available from http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf.
28. D. Veit, *Matchmaking in Electronic Markets*, Vol. 2882 of *LNCS*. Springer. Hot Topics, 2003.
29. W3C MathML, "Mathematical Markup Language (MathML) Version 2.0", 2003. Available from http://www.w3.org/TR/MathML2/.
30. W3C WSDL, "Web Services Description Language (WSDL) 1.1", 2004. Available from http://www.w3.org/TR/wsdl.
31. A.M. Zaremski and J.M. Wing, "Specification matching of software components", *ACM Transactions on Software Engineering and Methodology*, Vol. 6, No. 4, pp. 333–369, 1997.