Matchmaking Support for Mathematical Web Services

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

Abstract

Matchmaking is one crucial tasks in agent-based systems and describes locating and identifying the most suitable services among all available services. This paper introduces matchmaking of mathematical services, whereby the matching is based on semantics using OpenMath object descriptions of pre- and post-conditions. A matchmaking architecture for mathematical services is described containing four matching algorithms which achieve a structural match, a syntax and ontological match, an algebraic equivalence match and a value substitution match. The matchmaking architecture calculates the match scores which give an indication of the quality of the matches. A case study explains in detail how the matching process for all four matching algorithms works.

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 relevant 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 [8], LARKS [14], and IBROW [3]. 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 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:

$$T_{in} \geq C_{in} \wedge T_{out} \leq C_{out} \wedge T_{pre} \Rightarrow C_{pre} \wedge C_{post} \Rightarrow T_{post} \quad (1)$$

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 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 (Mathematics on the NET) [9] and GENSS (Grid-Enabled Numerical and Symbolic Services) [16] 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.

2 eScience Relevance

A significant number of applications within eScience make use of symbolic and numerical algorithms, developed as part of a project or obtained from third parties (such as numerical libraries from the Numerical Algorithms Group, or applications which use the Maple computer algebra system). The complexity of such algorithms can vary from simple matrix solving to more complex data analysis functions such as clustering or classification techniques. The ability to access such algorithms as Web Services allows easy integration of such capability within existing applications (while also providing a loose coupling between the application and the numerical algorithm). Existing eScience applications are still embedding symbolic or numerical techniques di-

rectly within an application. However, the ability to support such a loose coupling allows a user to select between multiple providers offering the same or a "similar" set of symbolic or numerical algorithms (services). A key driver of the work presented here is the ability to select between a number of possible algorithm implementations – assuming that the MONET approach has been adopted as the description technique for each algorithm. The approach therefore has usage in a number of possible application areas, and is not restricted to a particular scientific discipline. A "decomposition" service that enables a combination of algorithms to be combined to produce a similar results is currently being implemented, and makes use of mathematical reasoning techniques to support such decomposition.

3 Mathematical Matchmaking

3.1 Description of Mathematical Services

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 [17] 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 [17], 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.

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

3.2 Matchmaking Requirements

To achieve matchmaking:

1. 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

2. 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 [19]. 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 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.

3.3 Matchmaking Architecture

Our matchmaking architecture is shown in Figure 1 and comprises the following:

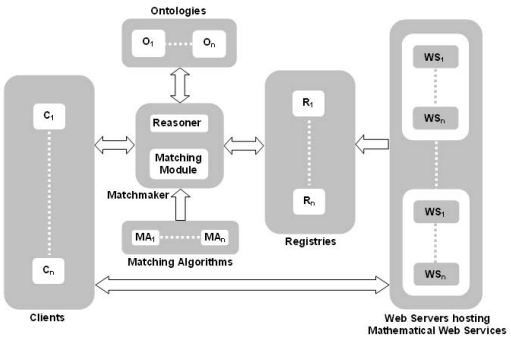

Figure 1: Matchmaking Architecture

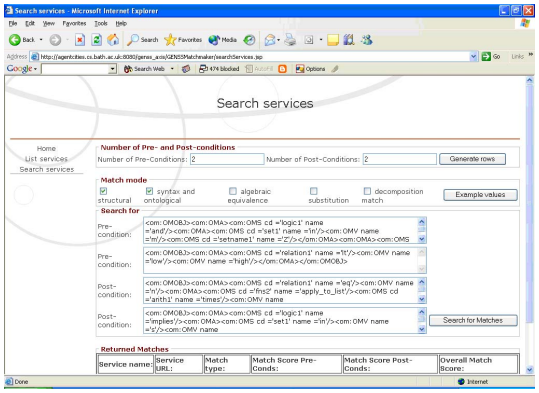

Figure 2: Matchmaker Client Application

- The client interface: this is employed by users to specify their service request.
- The matchmaker: this contains a reasoning engine and the matching module.
- Matching Algorithms: which define the logic of the matching process.
- Mathematical ontologies: including OpenMath Content Dictionaries (CDs), GAMS (General Algebraic Modeling System) *etc.*
- A registry service: which enables the storage of mathematical services.
- Mathematical Web Services: available on third party sites, accessible over the Web.

The interactions of a search request are as follows:

1. The user contacts the matchmaker.
2. The matchmaker loads the matching algorithms specified by the user via a lookup in the UDDI registry. In the case of an ontological match a further step is necessary. This is, the matchmaker contacts the reasoner which in turn loads the corresponding ontology.
3. Having additional match values results in the registry being queried, to see whether it contains services which match the request.
4. 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.

3.3.1 Client

The Client application[2] (shown in Figure 2) 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.

3.3.2 Matching Algorithms

Currently four matching algorithms have been implemented within the matchmaker. These are structural match, syntax and ontological match, algebraic equivalence match and value substitution match. Service descriptions defined in OpenMath allow descriptions of mathematical pre- and post-conditions. The structural match compares the OpenMath hierarchy at the *tag* level, without inspecting the attribute values. The syntax and ontological match algorithms go a step further and compare the `OMS` elements `cd` and `name` attributes values. The algebraic equivalence match and value substitution match perform mathematical reasoning on the mathematical objects which make up the pre- and post- conditions.

Structural Match The pre- and post-conditions are extracted and an SQL query is built to find the same OpenMath structure of the pre-/post- conditions of the service descriptions in the database.

Ontological Match This match is performed similarly, however the OpenMath elements are compared with an ontology representing the OpenMath elements. The matchmaking mechanism allows a more effective matchmaking process by using mathematical ontologies. Let us assume that the part of the ontology given by the CD setname1 satisfies: $\mathbb{C} \supset \mathbb{R} \supset \mathbb{Q} \supset \mathbb{Z} \supset \mathbb{N} \supset \mathbb{P}$. If the user query contains the OpenMath element:

```
<om:OMS cd='setname1'name='Z'/>
```

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

and the service description:

```
<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. We must note the implications given in equation 1, which imply that a queries pre-conditions must be less general than a capabilities (further to the 'right' in the above ontology). Whilst a queries post-condition must be more general than a capabilities (further to the 'left' in the above ontology). The above similarity value is $SV = \frac{1}{n} = 0.5$, where n is the degree of separation of the 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 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 ontologically and structurally different markup. Both are correct, it just depends on what information is required, so there can in general be no canonical form. In order to address the above observation, we must look deeper into the mathematical structure of the expressions which make up the conditions. Most of the conditions examined may be expressed in the form: $\mathrm{Q}(\mathrm{L}(\mathrm{R}))$ 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 terms, e.g. $=, \leq, \geq, \neq, \cdots$.

In most cases, the quantifier block will just be a range restriction. Sometimes it may be possible to use *quantifier elimination* to replace the quantifier block by an augmented 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 normal form we find useful for our matching technique is Disjunctive Normal Form. That is every logical block shall be converted into a Disjunction of conjunctions 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 valued, 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 Match 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 [11] 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.

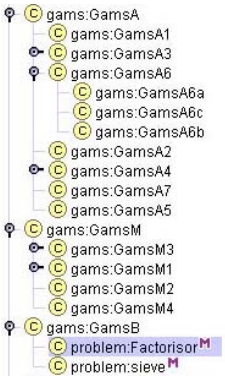

Figure 3: GAMS Taxonomy Fragment

Value Substitution Match 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.

3.3.3 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.

3.3.4 Mathematical Ontologies and Reasoning Engine

The subject of ontology is the study of the categories of things that exist or may exist in some domain [13]. An ontology is a catalogue of the types of things that are assumed to exist in a domain of interest from the perspective of a person who uses a language for the purpose of talking about a domain. The types in the ontology represent the predicates, word meanings, or concept and relation types of the language when used to discuss topics in the domain. An uninterpreted logic is ontologically neutral: It imposes no constraints on the subject matter or the way the subject is characterised. Logic alone says nothing about anything, but the combination of logic with an ontology provides a language that can express relationships about the entities in the domain of interest. The matchmaking mechanism which al-

lows a more efficient service discovery by using mathematical ontologies such as GAMS shown in Figure 3 are described in a semantic language and a reasoning engine can inference the ontology [7]. Used for the service discovery process was OWLJessKB [2]. It is intended to facilitate reading OWL files, interpreting the information as per OWL and RDF languages and allowing the user to query on that information. It then inserts these triples as facts into the JESS knowledge base [1]. With some predefined rules, JESS can reason about the triples and can draw more inferences. The JESS API (Application Programming Interface) is intended to facilitate interpretation of information of OWL files, and it allows users to query on that information. It leverages the existing RDF API to read in the OWL file as a collection of RDF triples.

3.3.5 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

$PrCQ \leftarrow read\_In\_PreConds\_From\_GUI()$
$PoCQ \leftarrow read\_In\_PostConds\_From\_GUI()$
$connect\_To\_DB()$
for $all\_service\_In\_DB$ **do**
  $PrC \leftarrow read\_PreConds\_From\_DB()$
  for $PrCS$ **do**
    $SVPrC \leftarrow select\_Match\_Algo()$
  end for
  $MVPrC \leftarrow calculate\_Match\_Value()$
  $PoCS \leftarrow read\_PostConds\_From\_DB()$
  for $PoCS$ **do**
    $SVPoC \leftarrow select\_Match\_Algo()$
  end for
  $MVPoC \leftarrow calculate\_Match\_Value()$
  $MVO \leftarrow calculate\_Match\_Score()$
  $SD \leftarrow retrieve\_Service\_Details()$
  $MD \leftarrow store\_Match\_Details()$
end for
$disconnect\_From\_DB()$
return MD

The matchmaking algorithm is specified in Algorithm 1. The pre- and post-conditions are read from the GUI 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 (2). 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 (3). The same is done with the post-conditions (4). The importance of the pre- or post-conditions is reflected in the weight values. The match scores may be calculated using the following equations:

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

$$M_A = \frac{w_a}{|A_Q| + |A_S|} * \frac{|A_Q|}{|A_S|} * \frac{\sum_{i=1}^{|A|}(SV_A(i)))}{|A|} \quad \text{where } 0 \leq w_a \leq 1 \tag{3}$$

$$M_B = \frac{w_b}{|B_Q| + |B_S|} * \frac{|B_Q|}{|B_S|} * \frac{\sum_{i=1}^{|B|}(SV_B(i)))}{|B|} \quad \text{where } 0 \leq w_b \leq 1 \tag{4}$$

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 $SV_A(i)$, $SV_B(i)$ are the similarity values for the ith matched pre-condition, post-condition respectively.

4 Case Study

For the case study we consider all 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 reflects 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:

$$\mathtt{n} = \prod_{i=1}^{l} \mathtt{lst\_fcts}_i \qquad \text{where } \mathtt{l} = |\mathtt{lst\_fcts}| \tag{5}$$

A user asking for a service with post-condition:

$$\forall i | 1 \leq i \leq |\mathtt{lst\_fcts}| \Rightarrow n \bmod \mathtt{lst\_fcts}_i = 0 \tag{6}$$

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

- equation (5) $\Rightarrow$ equation (6): This is clear since the value of n may be substituted into equation (5) and the resulting equality will be true for each value in lst_fcts.

- equation (6) $\Rightarrow$ equation (5): A slightly stronger version of equation (6) 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 5 and 6 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 $|\mathtt{lst\_fcts}| = \lceil \log_2(\mathtt{n}) \rceil$, 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 $\sqrt{\mathtt{n}}$.

- Then we calculate their product, from equation (5), this gives a new value for n.

- We may now check equation (6). 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 (5) and (6).

5 Related Work

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

The SHADE (SHAred Dependency Engineering) matchmaker [6] 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 [15]. Content languages of SHADE are a subset of KIF (Knowledge Interchange Format) [4] 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) [6] 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) [12] 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) [14] 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 [8] 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 [18] (Generic Request Architecture for Passive Provider Agents) 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.

6 Conclusion

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 plug-in matchers that may employ a variety of reasoning techniques, utilising theorem provers and computer algebra systems as well as information retrieval from textual documentation of mathematical routines. 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.

7 Acknowledgments

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] Java expert systems shell. `http://herzberg.ca.sandia.gov/jess/docs/61/index.html`.

[2] OWLJessKB. `http://edge.cs.drexel.edu/assemblies/software/owljesskb/`.

[3] V.R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel. Towards Brokering Problem-Solving Knowledge on the Internet. In Dieter Fensel and Rudi Studer, editors, *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW-99)*, volume 1621 of *LNAI*, pages 33–48, Berlin, May 26–29 1999. Springer.

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

[5] Mario Gomez and Enric Plaza. Extended matchmaking to maximize capability reuse. In Nicholas R. Jennings, Carles Sierra, Liz Sonnenberg, and Milind Tambe, editors, *Proceedings of The Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 144–151. ACM Press, 2004.

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

[7] S.A. Ludwig. Flexible semantic matchmaking engine. In *Proceedings of 2nd IASTED International Conference on Information and Knowledge Sharing (IKS), AZ, USA*, 2003.

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

[9] MONET Consortium. MONET Home Page, www. Available from `http://monet.nag.co.uk`.

[10] OpenMath Society. OpenMath website. `http://www.openmath.org`, February www.

[11] D. Richardson. Some Unsolvable Problems Involving Elementary Functions of a Real Variable. *Journal of Computational Logic*, 33:514–520, 1968.

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

[13] J.F. Sowa. Ontology, metadata, and semiotics, conceptual structures: Logical, linguistic, and computational issues. *Lecture Notes in AI #1867, Springer-Verlag, Berlin, pp. 55-81*, 2000.

[14] 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*, 5(2):173–203, June 2002.

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

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

[17] The OpenMath Society. The OpenMath Standard, October 2002. Available from `http://www.openmath.org/standard/om11/omstd11.xml`.

[18] D. Veit. *Matchmaking in Electronic Markets*, volume 2882 of *LNCS*. Springer, 2003. Hot Topics.

[19] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, October 1997.