

# Semantic Matching and Decomposition of Mathematical Services

William Naylor and Julian Padget  
Department of Computer Science  
University of Bath, UK  
{wn,jap}@cs.bath.ac.uk

Simone A. Ludwig and Omer F. Rana  
Department of Computer Science  
Cardiff University, UK  
{scmsal,scmofr}@cs.cardiff.ac.uk

## 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. We discuss the description of mathematical tasks and capabilities, processing descriptions for discovery and explore briefly how the descriptions enable task decomposition and capability composition.

If a service is to be found and used, it must advertise itself. There are many generic aspects of a service and several specific to mathematical services, but for the purposes of this abstract, we will concentrate on four, *viz.* the types of inputs and outputs and the pre- and post-conditions which specify the service's requirements and capabilities.

The role of the broker is to act as an intelligent mediator between clients and services, selecting services that match the client's problem statement (task). This task description must specify the signatures of inputs, outputs and the pre- and post-conditions that characterise the service that is sought. The broker's job is to identify service(s) or combinations of services satisfying the attributes given in the task. One of the major problems we address here is dealing with the alternative but equivalent representations that occur in mathematical descriptions of task and capability.

## 2. ONTOLOGIES FOR MATHEMATICAL SERVICES

We use various document structures defined in XML that have been developed in the MONET [1] project, while semantics are grounded using the OpenMath [2] extensible mathematical ontology representation mechanism. OpenMath supports the description of arbitrary mathematical objects via so-called Content Dictionaries (CDs) which define small specialised ontologies for different aspects of mathematics (currently <http://www.openmath.org> hosts some 50 CDs) and the markup associated with them. The MONET project has defined a number of ontologies which allow mathematical services to advertise their capabilities, describe mathematical tasks, return answers and explanations to tasks, etc. Mathematical Service Description Language (MSDL) builds on WSDL, defining the signatures of inputs and outputs the pre- and post-conditions, the content of each of which is typically OpenMath<sup>1</sup>

## 3. NORMAL FORMS FOR MATHEMATICAL OBJECTS

Many mathematical objects do not have unique descriptions: for example if  $i, j \in \mathbb{Z}$ , then  $i \leq j$  and  $i - 1 < j$  are equivalent. This implies that the descriptions of mathematical capabilities in general may not be unique, and consequently creates a significant problem for the broker in identifying appropriate capabilities, as the descriptions must necessarily often contain complex mathematical objects. This is the problem we currently face and we will now describe how we are tackling it. We have observed that most expressions take the form of  $Q(L(R))$  where:

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

Despite the fact there can be no absolute normal form, we can nevertheless carry out a sequence of transformations to construct a normal form suited to our brokerage task, thus:

1. Associative operators have the property (for operation  $\otimes$ ) that  $a \otimes (b \otimes c) = (a \otimes b) \otimes c$ . Because of this we may get expressions which are nested in different ways. A natural normalisation of this is to represent it as an  $n$ -associative operator and *flatten* the arguments, so that  $a \otimes (b \otimes c) \rightarrow \otimes(a, b, c)$  and  $(a \otimes b) \otimes c \rightarrow \otimes(a, b, c)$ .
2. The logical parts of the task and capability are rewritten in Disjunctive Normal Form (DNF), which is convenient for the calculation of similarity values (see §4)
3. The numerous domain specific mathematical equivalences that exist are addressed by means of a database of rewrites, where the right hand side is the normal form.
4.  $\alpha'$  conversion<sup>2</sup> addresses the situation where there are bound variables in the description as a result of quantification or a differential operator (or other). There is no reason for task and capability names to coincide, but a straightforward transformation ensures they do, making subsequent manipulations simpler.
5. Commutative operators offer opportunity for confusion like associative since there will be a number of ways to represent an expression using them. Our solution is to define an ordering on the elements of OpenMath objects, then when a capability is registered, the children of any commutative operations will be stored in order. A similar sorting is performed on the task. As long as the sorting is structurally based this means that regardless of the ordering (as long as it is well founded), capability and task will be identical down to their leaves. However there is a problem with identifying variables which are direct descendants of a commutative operation; we deal with this by constructing equivalence classes of documents that are structurally identical modulo variable differences.

<sup>1</sup>Other languages are possible, such as Content MathML, but we use OpenMath for extensibility and our own familiarity.

<sup>2</sup>By which we mean consistent variable renaming.

## 4. SIMILARITY MEASURES

Once the task description has been normalised, it can be compared with the capability descriptions registered with the broker, with the objective of calculating a similarity value. We denote the pre- and post-conditions of task and capability descriptions by  $T_{pre}, T_{post}, C_{pre}, C_{post}$  and express the matching requirement between them as:

$$T_{pre} \Rightarrow C_{pre} \wedge C_{post} \Rightarrow T_{post} \quad (1)$$

That is to say, the pre-conditions of the capability must be satisfied by the pre-conditions of the task and the post-conditions of the task must be satisfied by the post-conditions of the capability. 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 capability pre-condition. Superfluous capability pre-conditions (task post-conditions) do not effect whether the function may be performed. However it is necessary that there are no extra task pre-conditions (capability post-conditions) as this might allow the client to provide conditions incompatible with the capability pre-conditions or post-conditions. This may be formalised in the following:

$$\exists x_1 \in C_{pre} \text{ s.t. } \forall y_1 \in T_{pre} \mid y_1 \Rightarrow x_1 \quad (2)$$

and

$$\exists x_2 \in T_{post} \text{ s.t. } \forall y_2 \in C_{post} \mid y_2 \Rightarrow x_2 \quad (3)$$

We shall treat the pre- and post- conditions separately to get two similarity values  $S_{pre}$  and  $S_{post}$ . These should both contribute equally to the similarity value between a task and a capability, as both expressions 2 and 3 must be satisfied if there is a match. So we may calculate a value by averaging  $S_{pre}$  and  $S_{post}$ . We shall denote the DNF for  $C_{pre}$  (or  $T_{post}$ ) by  $R_1 \vee \dots \vee R_n$  and for  $C_{post}$  (or  $T_{pre}$ ) by  $S_1 \vee \dots \vee S_n$ . To calculate a value  $\in [0.0, 1.0]$  indicating how well equations 2, 3 are satisfied, we use the formula:

$$similarity = \max_{i=1 \dots n} \{ \min_{j=1 \dots n} \{ M(R_i, S_j) \} \} \quad (4)$$

where  $M(R_i, S_j)$  calculates how well conjuncts  $R_i$  and  $S_j$  match. We may calculate a value for  $M(R_i, S_j)$  as:

$$\sum_{k=1 \dots \delta} m(S_j, R_{i,k}) \frac{1}{\delta} \quad (5)$$

where  $\delta$  is the number of terms in  $R_i$ ,  $R_{i,k}$  are terms in  $R_i$  and:

$$m(S_j, R_{i,k}) \text{ returns } \begin{cases} 1.0 & \text{if } R_{i,k} \text{ matches a term in } S_j, \\ 0.0 & \text{otherwise.} \end{cases}$$

More complex matching functions may be conceived e.g. a term matching its negation could cause a penalty to be subtracted from the similarity value.

## 5. AN APPROACH TO AUTOMATIC SERVICE COMPOSITION

The DNF representation of capabilities also enables a constructive approach to service composition. For clarity we shall assume the pre-conditions and post-conditions of a query and the services are trivial disjuncts, i.e. a single conjunct. We observe that a task which has pre- and post- conditions  $T_{pre}$  and  $T_{post}$  may be satisfied by two capabilities  $C_1$  and  $C_2$  with pre- and post- conditions  $C_{pre1}, C_{post1}$  and  $C_{pre2}, C_{post2}$  respectively if:

$$T_{pre} \implies C_{pre1} \quad (6)$$

$$T_{pre} \wedge C_{post1} \implies C_{pre2} \quad (7)$$

$$C_{post1} \wedge C_{post2} \implies T_{post} \quad (8)$$

by executing  $C_1$  followed by  $C_2$ . This basic approach may be extended to automatic service composition for many services.

The algorithm required for calculating the similarity value for conjuncts given by expression (5) requires determining a match between individual terms, this allows us to determine which conditions have and which conditions have not been satisfied. A greedy approach may be used to discover some composition of services where overall the task post-condition is met and whose pre-conditions may be satisfied by a conjunct of the task pre-conditions and post-conditions of other services.

## 6. EXAMPLE

An example of a task executed by two services, which displays our architecture for Automatic Service Composition, is the following:

**input:** a set of points  $(pt_i, val_i)_{i=1 \dots n} \in (\mathbb{Z} \times \mathbb{Q})$   
a degree  $d \in \mathbb{Z}$  with  $d < n$

**output:** a value  $R = \int_{pt_1}^{pt_n} f(x) \wedge f \in \mathbb{Z}[x] \wedge \text{degree}(f) = d$   
where  $f(pt_i) = val_i$  for  $i = 1 \dots n$

This task may be executed using the composition of the following two services:

### Interpolation service

**input:** a set of points  $(pt_i, val_i)_{i=1 \dots n} \in (\mathbb{Z} \times \mathbb{Q})$   
a degree  $d \in \mathbb{Z}$  with  $d < n$

**output:** a polynomial, with degree =  $d$   
and  $f(pt_i) = val_i$  for  $i = 1 \dots n$

and

### Definite Integration service

**input:** a function  $f: \mathbb{R} \rightarrow \mathbb{R}$   
a lower and upper bound  $l, u \in \mathbb{R}$

**output:** the definite integral  $\int_l^u f(x) dx$

## 7. CONCLUSION

What we believe is interesting and novel about the work presented here is not so much the conditions for service matching, which are fairly conventional, but:

- their application in the context of mathematical web service discovery, which is a semantically rich and challenging domain and as far as we can tell less amenable to the subsumption-style matching commonly used in brokerage
- the techniques for the normalisation of mathematical semantic descriptions
- the techniques for composition of mathematical services
- and the integration of OpenMath with the service matching process.

## 8. REFERENCES

- [1] Mathematics on the net - MONET.  
<http://monet.nag.co.uk>.
- [2] The OpenMath Society. The OpenMath Standard, October 2002. Available from <http://www.openmath.org/standard/om11/omstd11.xml>.