Description Schemes for Mathematical Web Services

M. Dewar & D. Carlisle
NAG Ltd, Oxford, UK

O. Caprotti
RISC, Linz, Austria

Abstract

While commodity technologies are now available for deploying and accessing web (and grid) services, the issue of how a potential user discovers a service which matches his or her requirements is still open. This paper looks at the specific case of mathematical web services and proposes two complementary solutions to this problem.

1. INTRODUCTION

In this paper we discuss some proposed solutions to the problem of describing the domain of applicability of a mathematical web service, by which we mean a web service which is designed to solve some class of mathematical problems. Some of these services might have very broad functionality, such as "calculus" or "linear algebra", while others might implement a single very specific algorithm.

The process of *Web Service Discovery* consists of finding the best match amongst a collection of available services to a particular problem or kind of problem. In our particular area of interest, mathematics, this matching process will involve both mathematical ("can it solve my problem?") and non-mathematical ("am I authorised to use it?", "will the resources needed to solve my problem be available soon?", ...) criteria, but this paper is only concerned with the former.

In the web service community, XML languages have been proposed to describe a service ([WSDL](#)) and to publish a service ([UDDI](#), [WSIL](#)), and these are now beginning to be used in e-business. From our own investigations however, these approaches are not suitable for capturing the complete nature of a mathematical service. [UDDI](#) for instance is a taxonomy-based approach where the taxonomy is on the kind of service provided and would collapse all mathematical service providers under one category, which does not help to facilitate the discovery of a service suitable for handling a specific mathematical problem. [UDDI](#) does have mechanisms for registering external, more specific taxonomies, see for example [\[Providing a Taxonomy For Use In UDDI Version 2 \[3\]\]](#), and it may eventually be possible to integrate specific information about mathematical services into a general scheme such as [UDDI](#), however at this stage we focus on the specific issues relating to classifying mathematical services. The facilities for extending [UDDI](#) with application-specific information have improved with successive [UDDI](#) versions and it seems inappropriate to focus too much on the details of the extension functionality available in the current [UDDI](#) draft. [WSDL](#) on the other hand allows a service provider to describe the low-levels details of the service interface which is useful, but not enough for our purposes: a mathematical service has a natural formal description capturing its behaviour which is independent, say, of the actual data types used in the implementation.

2. OPENMATH AND MATHML

[OpenMath](#) [\[1\]](#) is a standard for representing mathematical data in as unambiguous a way as possible. The

original motivation for OpenMath came from the Computer Algebra community, where packages were getting bigger and more unwieldy and it seemed reasonable to adopt a generic "plug and play" architecture to allow specialised programs to be used from general purpose environments. While there were plenty of mechanisms for connecting software components together, no common format for representing the underlying data objects existed. OpenMath can be used to exchange mathematical objects between software packages or via email, or as a persistent data format in a database. It is tightly focused on representing semantic information and is not intended to be used directly for presentation, although tools exist to facilitate this via LaTeX, MathML etc. It is extensible, and its most common representation is as XML.

OpenMath works by cataloguing the semantics of symbols in *Content Dictionaries (CDs)*, each of which covers a quite narrowly-defined area of mathematics. All symbols in OpenMath objects are attributed with the name of the CD to which they belong, and new CDs can be written by anybody at any time. The OpenMath language contains a grammar which describes how the symbols can be combined, and how primitive objects such as integers, strings etc. can be constructed. OpenMath can be regarded as an ontological framework while any collection of CDs forms an ontology.

MathML [2] is a recommendation from W3C, dealing both with presentation of mathematical objects and (within a limited range) their semantics or "Content". The content elements may be attributed with a URI describing their semantics, but MathML provides no standard mechanism for encoding that description. In particular this means that it is possible to use the MathML syntax but describe the semantics of each element by reference to an OpenMath CD. In this framework though we expect that Mathematical semantics will be expressed in XML using the OpenMath encoding, with (Presentation) MathML being used by some services that need to provide platform independent rendering of mathematical objects. In those cases where it is desirable to use Content MathML syntax, perhaps to take advantage of existing support for that language, it is relatively straight forward to define a mapping between the two languages. The details are given in the paper [4].

In the rest of this paper when we refer to "web services" we will mean mathematical web services which use OpenMath to encode their data and results. Taking the definition given in the W3C Working Draft on Web Services Architecture Requirements [5]:

Definition: A *mathematical web service* is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts and whose data and results are encoded using OpenMath. A mathematical Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.

Since OpenMath Content Dictionaries are quite specific, it is convenient to describe a web service's precise input and output languages in terms of the CDs which it can refer (e.g. via a suitable Schema). These languages are not necessarily the same: for example the input to a service performing symbolic integration might well include symbols from the calculus1 CD such as `int` and `defint` (for indefinite and definite integration respectively), but the output would not.

3. THE TAXONOMY APPROACH

A simple approach to describing a web service is by defining the language that it uses to interact with users and other services, and identifying what it does with its data by reference to a fixed taxonomy. We have already mentioned UDDI which we believe is inappropriate for mathematical queries. George Polya, in his book "How to Solve It" [9], identified two broad classes of problems: problems to find and problems to prove. In a "problem to find", the main objective is to find the unknown given the data and conditions. In a "problem to prove", the aim is to show the validity of a certain assertion, the conclusion, under given hypotheses. Each class of problem is described in a different way (although it is possible to state a "problem to prove" as a "problem to find" in Type Theory, but restricting to one formal system is too limiting.)

Within one class of problems in such a broad classification, one may look at a fine-grained taxonomy. For instance, if we consider only "problems to find", a very rich taxonomy is the Guide To Mathematical Software [6] produced by the National Institute For Standards And Technology. This catalogues available software packages according to the problems they can solve, and is organised in a hierarchy so that, for example, the area of differentiation and integration has identifier "H", while the automatic solution of a one-dimensional

definite integral over a finite range where the integral is defined procedurally has identifier "H2a1a1". Note that the latter is still not precise enough to identify a specific algorithm: for example the NAG Fortran Library currently contains five algorithms in this area.

It is debatable whether it makes sense to deploy single algorithms as individual web services for solving this kind of integral and so in this case this level of granularity is probably enough. However there are certainly instances where a single very specific algorithm could be deployed as a web service in its own right. At present, the GAMS classification is skewed very much towards numerical software so for example there are no subdivisions of symbolic computation ("O"), meaning that one cannot even distinguish between a package like GAP for discrete algebra and Maple for traditional computer algebra.

Another feature of GAMS is that the problem areas are not mutually exclusive, and it is possible for a particular problem or piece of software to belong to several. For example optimisation is category G, but also turns up under K (approximation) and L8 (regression). The NAG routine E04USF, which computes the minimum of a sum of squares subject to non-linear constraints, belongs to all these categories and appears in particular in G2h2a1, K1b2b and L8b1b2.

Despite these drawbacks we believe that GAMS is the best starting point currently available and quite suitable for our purposes. In this scheme the process of service discovery may take two different forms:

1. The client can itself identify which category of service it needs and look it up in a suitable registry (either filtering out those with which it does not share a common set of understood content dictionaries, or selecting a suitable translation service to act as an intermediary).
2. The client can send its problem to an agent whose task is to classify it, and then select a suitable service.

4. THE BEHAVIOURAL APPROACH

A different approach to using a fixed taxonomy is to provide a behavioural specification of the interface to the service, along with side-conditions on the input and the output. We would like to stress that this is made possible by the nature of the tasks performed by mathematical services. For example the description of an optimisation service might be as follows:

- Input:
 1. $F: \mathbf{R}^n \rightarrow \mathbf{R}$
 2. $A \subseteq \mathbf{R}^n$
 3. $\epsilon \in \mathbf{R}$
- Output:
 1. $x \in \mathbf{R}^n$
 2. $f \in \mathbf{R}$
- Conditions:
 1. $F(x) = f$
 2. $\exists y \in A$ such that $F(y) < f - \epsilon$

Notice that each of the mathematical statements given above can be expressed in XML using OpenMath. While this is similar to a formal definition of a GAMS classification it has two major differences:

- a. it refers to specific objects in the interface;
- b. it allows for a much tighter definition of the circumstance under which the algorithm applies.
- c. it allows for reasoning on generalisations or specialisations of the problem solved

So for example we could add an extra input:

$$4. \quad D: \mathbf{R}^n \rightarrow \mathbf{R}$$

along with an extra condition:

$$3. \quad D = F'$$

to indicate that the service requires the client to provide not just the objective function but its derivative. We could also add conditions to indicate whether the constraints which define the region A in which the minimum lies are linear or non-linear etc.

It is clear that this is a much more flexible approach than the taxonomy described earlier, but it does have some drawbacks. Identifying a suitable service for solving a particular problem becomes much harder because either the formal descriptions of both have to match *exactly*, or one has to be able to prove that they are logically equivalent, or one has to be able to prove that the problem is a special case of the service's specification.

Automated deduction techniques have to be used to carry out the reasoning involved in the matching process described above. Positive results in the area of proof planning in which high-level proving strategies are searched and matched against each other provides evidence that this approach can be implemented in an effective way (see for example the Omega system).

5. DISCUSSION

On the face of it the taxonomy approach is much easier for both clients and service providers than the behavioural one. If one knows what kind of problem (e.g. optimisation) one wishes to solve, then it is simply a matter of looking it up in an appropriate registry. While static registries are very difficult to maintain (the GAMS description of the NAG Fortran Library [7] is, at the time of writing, over a year out of date; and the UDDI Weather Report [8] recently found that only 43% of UDDI entries in a major registry were correct), it ought to be possible to build registries dynamically in much the same way that search engines try to maintain a current view of the web (see for example, WSIL).

Taxonomies cannot, however, remain static, and must evolve to reflect new services as they become available. This poses problems for service providers who must ensure that the categorisation of their services is up to date, as well as for clients. The GAMS classification goes into a great deal of detail in some areas but is far too broad in others.

The behavioural description on the other hand does allow service providers to publish fixed descriptions of their services. However the mapping from a problem to the description is much more complicated. We mentioned earlier that a particular NAG routine, E04USF, can be used to solve problems in optimisation and regression. However from a client's point of view these look very different: an optimisation problem is defined by a function, a set of constraints and some initial guess as to the point at which the solution occurs, whereas a regression problem is defined by a set of points and some information about the structure of the underlying function. When using the taxonomy approach, one expects the service provider to provide a binding that relates the taxonomic problem description with the service interface. When using the behavioural description the software doing the matching process needs some extra domain-specific knowledge to re-formulate the user's problem (in this case re-pose the regression to minimising the residuals in a sum of squares). Because the services we deal with handle mathematical objects rather than general data and have a very precise formal behavioural description, we can take advantage of this situation if we reason about the abstract definitions.

Where the behavioural approach is strongest, however, is in matching the features of a particular problem to a particular algorithm or group of algorithms. This implies that either the software doing the matching can do a certain amount of mathematical analysis or can ask that the client or another web service does it on its behalf. For example, the choice of numerical integrator may be determined by whether the integrand has singularities and, if so, what form they take. Mathematical service discovery in this sense will be carried out by specialised brokers dealing with a specific mathematical area that are able to find, given a query, the most appropriate service, or to suggest a problem-solving strategy in the form of a sequence of services.

6. CONCLUSIONS

While UDDI may be suited to business-to-business applications we do not believe that it captures enough detail to facilitate the discovery of scientific and, in particular, mathematical web services.

An alternative taxonomy, such as that offered by GAMS, is much more appropriate. However the level of detail in some areas is excessive for our purpose while in others it is insufficient. Aiming for too fine a granularity would lead to instability in the taxonomy, i.e. it would change too frequently. Also one only requires the taxonomy to be fine enough to distinguish the services that are being provided.

Taking the more formal behavioural approach on the other hand requires a lot of extra computational machinery and makes the deployment of a web service more complicated. However it does allow for a very precise definition of the domain of applicability of a web service to be published and reasoned about.

We therefore propose a combined approach, which we are investigating as part of the the MONET Project. We will adopt a taxonomy based on GAMS which is relatively coarse grained, for example in the integration area quoted earlier it would stop at "one dimensional quadrature" rather than going down a further three or more levels. This can be viewed as shorthand for a formal description. Services which wish to advertise a more refined applicability can provide a more precise behavioural description which can be viewed as restricting them to a subclass or special case of the problem area. This approach makes the process of deploying and describing web services relatively straightforward while not ruling out the use of more sophisticated agent-based systems to match problems to solution strategies.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the MONET Project (IST-2001-34145) and the OpenMath Thematic Network (IST-2000-28719), both sponsored by the European Union. In addition Olga Caprotti is supported by the Austrian Science Foundation (FWF) (Project P15183).

REFERENCES

- [1] The OpenMath Standard. The OpenMath Consortium, 2000.
- [2] Mathematical Markup Language (MathML) Version 2.0 W3C Recommendation, 2001.
- [3] Providing a Taxonomy For Use In UDDI Version 2, UDDI Working Draft Technical Note Document. July 17, 2001.
- [4] Conversion between MathML and OpenMath, OpenMath Consortium deliverable, 2001.
- [5] Web Services Architecture Requirements, W3C Working Draft 19 August 2002.
- [6] Guide to Available Mathematical Software, NIST, 2002.
- [7] GAMS Package, NAG, NIST
- [8] UDDI Weather Report, SalCentral, 2002
- [9] "How to Solve It", 2nd ed., G. Polya, Princeton University Press, 1957, ISBN 0-691-08097-6.

© O. Caprotti, D. Carlisle, M. Dewar 2002