Description and generation of mathematical web services

Marc-Laurent Aird, Walter Barbera Medina, James Davenport, Julian Padget
Department of Computer Science
University of Bath
BATH BA2 7AY, UK
{mapma,wbm,jhd,jap}@cs.bath.ac.uk

ABSTRACT

The evolution of the web into a repository of documents and resources for *programs* to read instead of humans and in which programs are published as services and invoked through messaging offers some interesting technical challenges, as well as opportunities, for mathematical software. The aim of the MONET [11][1] project is a proof-of-concept, building on the standards defined so far by W3C, to demonstrate that (a) mathematical web services can be simply constructed and deployed (b) precise descriptions of mathematical services can be written and published (c) basic brokerage facilities can be built identifying services that match the problem (d) advanced brokerage facilities can be built orchestrating several web services to solve a problem In this paper we focus on the first two aspects and provide a brief outline of the second two, all illustrated by reference to services involving the NAG library and the Maple computer algebra system.

1. INTRODUCTION

Currently, the web is a vast collection of information stored in a variety of formats and held loosely together by hyper-links. This is fine for human surfers who are able to filter out excess noise and understand natural languages, but it is no use if we want machines to be able to process and understand information. The next step is the semantic web [1], where information is encoded according to well-defined vocabularies that are designed to make the meaning of data explicit and unambiguous. The development of this vision will make it possible to automate a wide variety of sophisticated interactions, which today require human intervention.

The aim of the MONET project is to demonstrate the applicability of the semantic web to the world of mathematical software, using sophisticated techniques from AI and mathematics to match the characteristics of a problem to the advertised capabilities of available services and then invoking the chosen services through a standardised mechanism. The resulting framework is intended be powerful, flexible and dynamic, yet robust and easy to navigate, putting state-of-the-art algorithms at the disposal of users anywhere in the world.

This paper discusses the MONET architecture and describes the work carried out to date on building a demonstration symbolic mathematics web service. For the Bath demonstrator we have chosen to use Maple as the computer algebra system, but in principle any system could be used in the same way, for example, Reduce, Axiom, GAP, Magma. The key requirement is support for OpenMath

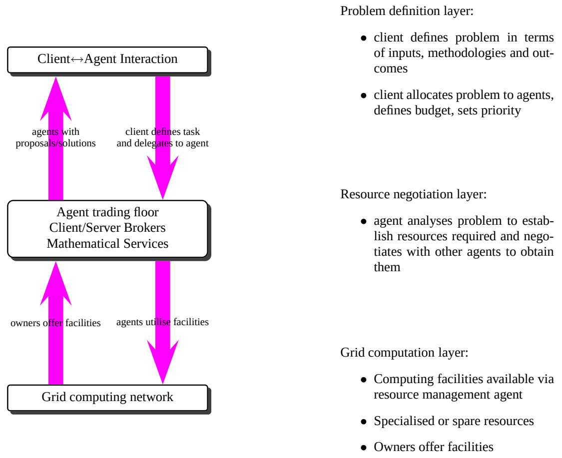

Figure 1: MONET and the Grid

or MathML input and output, although this functionality can be achieved by wrapping the algebra system within a phrasebook that carries out translation between these network-neutral representations and the system's presentation language. A similar symbolic service based on Maple has also been developed at the University of Western Ontario (UWO), while complementary numerical service has been developed by the Numerical Algorithms Group (NAG) at Oxford [13]. We begin with a brief overview of the MONET architecture, before taking a detailed look at service description and the current demonstration service, explaining how it is invoked, and the types of information passed through the service and across the web. In its completed state such a service will be discovered dynamically by a broker or planner by matching client requirements and problem descriptions to properties of advertised services.

The client, broker and each service will describe mathematical objects using OpenMath [12, 3], an extensible semantic format for unambiguously describing mathematical objects and properties, embedded within Mathematical Service Description Language (MSDL) documents, which extend the information currently provided by Web Service Description Language (WSDL) documents. OpenMath describes mathematical concepts by way of symbols defined in Content Dictionaries (CDs). For example, the function $\sin$ is found in the *transc1* CD of transcendental functions. However, in keeping with W3C conventions, we do not mandate the use of any particular technology except XML, so although we write here about using OpenMath and WSDL, these could equally well be MathML and IDL, for instance.

[1]The MONET project (`monet.nag.co.uk`) is funded by the Commission of the European Communities Information Society program (project IST-2001-34145)

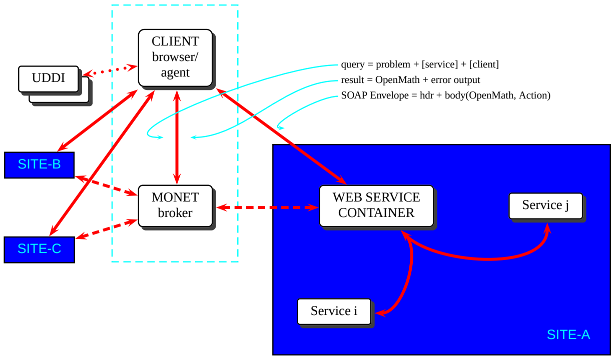

Figure 2: General Monet Architecture

MONET can be viewed as complementary to the GRID by building a problem-oriented application layer on top of grid services, in which tasks are decomposed for distribution to specialised mathematical services, which themselves are in turn supported by grid-provided resources, as suggested in Figure 1.

1.1 MONET Architecture Overview

The general MONET architecture is sketched in Figure 2. When the user submits a problem to a MONET broker (note that there is no notion of a centralized brokerage service: MONET brokers are expected to be instantiated on a per-client basis, as shown here), they may choose either then or later to delegate service negotiation, selection and invocation to the broker, or they may simply use the broker as an intelligent search engine which will return a collection of available services or even an execution plan choreographing multiple services.

In respect of our demonstrator, most code has been written in Java, and services have been deployed using AXIS [10]. In order to meet our requirement of using OpenMath as the content language in client-server communication, we wrote a Maple phrasebook to translate OpenMath into Maple's input language. This Java phrasebook has been developed using the RIACA OpenMath Library [9] and mainly covers symbols from the MathML CD Group [12]. It has been made into a web service too and is available through `http://agentcities:8070/axis/services/OM2MapleTranslationService?wsdl`. Fortunately, Maple has the option of generating MathML output and it is relatively simple to transform that into OpenMath when returning the result of the service.

2. SERVICE DESCRIPTION

In MONET terms a service is defined by the combination of its Mathematical Services Description Language (MSDL)[2] document which contains mathematical information about the type of problems the service solves and logistical information (software-dependent) such as the invocation protocol, cost, etc. We expect that a Maple developer would be required to complete the mathematical parts of the MSDL, while much of the rest would be static across all such symbolic services regardless of the underlying computer algebra system. A Mathematical Service Description consists of the following (note how the MSDL provides an extended service description intended to wrap around WSDL):

Classification: specification of what the service does

- Reference to a problem description library. This consists of a *header*, containing a bibliographic entry and pointer to an equivalent problem in some taxonomy, and a *body*. The body describes the type of problem a service solves by specifying inputs, outputs, post- and pre-conditions. This form of problem description is also re-used in the client's query.
- Reference to taxonomies. This is the easiest way to describe what a service does. It is intended that MONET will develop a taxonomy based on GAMS (Guide to Available Mathematical Software) [7], in particular to include sub-categories under the current all-inclusive "Symbolic Computation" node.
- Supported Semantics. For example this could be a list of OpenMath CDs supported. If the CD *calculus1* is listed then the service could be a candidate for solving integration problems. Likewise, if the *minimax1* CD were on the list, then the service could maybe handle optimisation problems.
- Supported Directives. These are verbs used to describe the type of action a service performs, and also would form part of the client query. Examples include *solve*, *prove* and *decide*.

Implementation Details: information about the specific service

- A reference to an Algorithm Description. Again, an OpenMath CD, containing symbols for describing algorithmic complexity, will be produced.
- Software Details
- Hardware Details
- Algorithmic Properties. These could include such properties as accuracy and resource usage.
- Descriptions of actions needed to solve a problem. These are used in the case where a service requires several steps to perform a computation and could be such actions as *initialise*, *exec*, *stop*. These actions are also included in the Service Binding description below, where they are mapped into concrete operations.

It should be noted that not all of the Classification or Implementation details are mandatory.

Service Interface Description: This would typically be a WSDL document.

Service Binding Description: This is a mapping from abstract problem components and actions to elements of the service interface, thus allowing calls to the service to be constructed.

Broker Interface: This is the API exposed to the broker by the service allowing it to generate service instances, perform housekeeping actions etc. Typically it would consist of a service URI and an interface description.

3. DEPLOYMENT

Some technical details aside, deploying a program as a web service is relatively straightforward. Having done a few, it rapidly becomes apparent that much of the necessary software glue is common to many services and so we have sought to take advantage of that in developing a Maple wrapper service, which accepts a piece of Maple code and then deploys it as a new service on that server. It has also provided us with the experimental context for the refinement of MSDL as we have tried to deploy a range of mathematical services. It should be noted that there is nothing specific to Maple or even computer algebra about this.

3.1 Maple Wrapper Service Architecture

We outline the process of submitting and invoking Maple code in Figure 3. The Maple wrapper service fulfils two functions: (i) receiving some Maple code and exposing it as a MONET service by means of registering an MSDL document, (ii) enabling that code to be executed remotely by a client. We now discuss these in more detail.

First, a Maple developer contacts the web service, passing it some

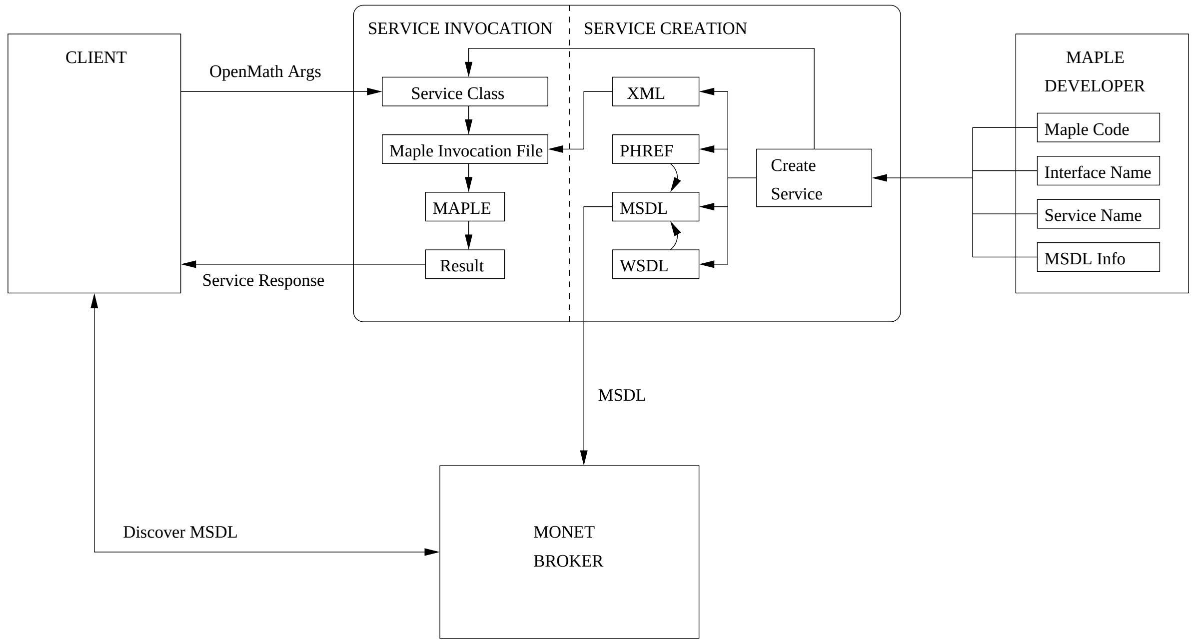

Figure 3: Architecture of the Maple Wrapper Service

Maple code, a name for their service, a Maple procedure name identifying the entry-point, and some MSDL information specific to their code and the type of problem it relates to. From this the service creates an XML file on the server containing invocation information such as the Maple code. An MSDL file is also created containing a reference to a problem description file in terms of inputs, outputs, pre-conditions and post-conditions.

To invoke the code, a client provides the service name and some appropriate mathematical arguments, written in OpenMath, which correspond to the inputs in the problem description. The Web service creates a temporary invocation file which reads in the Maple code from the XML file, and invokes it using the procedure name provided by the Maple developer together the arguments passed by the client. This temporary file is executed in Maple from the command line, and the result returned to the client.

It should be noted that there is essentially only one actual web service which stores mathematical descriptions and executes Maple algorithms. Each Maple service created results in a different MSDL file which can be registered with a service discovery broker. Although each MSDL contains its own WSDL describing how to access the web service, the latter is just a mask, as all Maple services end up invoking our wrapper service. That is to say, the submission of each mathematical problem description can be thought of as a Maple service which can be executed through its own web service, although it is not strictly a web service in its own right as they all invoke the MapleWrapper Service. This should become clearer as we work through some examples in Sections 3.4, 4.1 and 4.2.

We now go on to describe the MSDL parts we expect a Maple developer to provide, which parts we expect to be provided by the Web service, and how these are described in the MSDL document. Typically, problem specific information will be provided by the Maple developer, while the web service will provide information which will be static across all such Maple services and possibly most computer algebra services.

3.2 Maple-Developer Provided Information

We expect the Maple developer to submit the following information in order to create the Maple service.

- Some Maple code.
- An interface name used to invoke the Maple code.
- A service name.
- A Mathematical Problem Description.
- A taxonomy reference.
- A list of supported directives.
- A list of algorithm references.
- A description of the algorithmic properties.

We illustrate this information by way of a numerical integration example. This example is equivalent to the numerical service described in [13].

3.2.1 Maple Code and Interface Name

Suppose a Maple developer writes the a procedure for numerical integration using the Gauss Legendre method [5, 8] and wishes to expose it as a web service (this example and all the relevant code is in [6]). Firstly, that Maple code will be stored in a file. Secondly, the Maple developer submits the name of the procedure which executes the service, the so-called interface name, which in our case happens to be `gausleg_int1`. Finally, we specify each argument of this procedure: a function, a variable, the upper and lower bound of the range of integration, and a desired level of accuracy, through variable names and OpenMath descriptions. A legitimate Maple invocation of such procedure could be:

```
gausleg_int1(1/(1+xˆ2), x, -1, 1, 0.001);
```

In addition we need to specify the return values: a list consisting of the numerical approximation to the solution, plus a bound on the accuracy of that solution. For this reason, we require from the Maple developer a complete problem description in terms of inputs, outputs, pre- and post-conditions, as described in Section 2.

3.2.2 Mathematical Problem Description

The purpose of the MSDL document is twofold: (i) To provide a client with information about how to invoke a particular service, (ii) To permit a broker to match a client's query to a service which could solve it. The mathematical problem description is thus a very important part of the service description as it can be used by the broker when choosing an appropriate service, and by the client to see the types of the input arguments the service accepts. Consequently it is crucial that we have a problem description in OpenMath of the inputs, output, pre- and post-conditions. While this general concept is fairly well understood in formal software engineering, its application to the definition of mathematical services is novel and needs much more work. If the service describer is unfamiliar with OpenMath – and since the notation is fairly human-unfriendly – then it might perhaps be practical to make use of a phrasebook for translating from a language with which they are familiar into OpenMath. An example of a problem description for the numerical integration service in question is:

Input:	1. $F : \mathbb{R} \rightarrow \mathbb{R}$ 2. $a \in \mathbb{R}$ 3. $b \in \mathbb{R}$ 4. x 5. $e \in \mathbb{R}$
Output:	1. $I \in \mathbb{R}$ 2. $E \in \mathbb{R}$
Post-Condition:	1. $E < e$ 2. $\lvert I - \int_a^b f(x)dx \rvert < E$

A corresponding MSDL problem description encoded in XML is listed in the appendix to [6]. The problem description will normally be located in a separate file and referenced from the MSDL file, for example:

```
<monet:classification>
  <monet:problem href="http://www.cs.bath.ac.uk/
   ˜mapma/Monet/num_int_problem_desc.xml"/>
</monet:directive-type>
```

3.2.3 Other MSDL entries

We now describe the other elements of the MSDL which we expect the Maple developer to provide.

Taxonomy Reference The example above corresponds to numerical routines listed in the `H2a1a1` category in the GAMS classification, namely *Automatic 1-D finite interval quadrature (user need only specify required accuracy), integrand available via user-defined procedure*. The following snippet would be included in the MSDL document to describe this reference:

```
<monet:classification taxonomy=
 "http://gams.nist.gov/" code="H2a1a1"/>
```

It is expected that the Maple developer would choose which node(s) in the taxonomy best match their proposed service by choosing from a published list. We may also have a taxonomic reference within the Algorithm Reference element as shown below.

Directive Directives are the verbs which describe what types of actions a service performs. The numerical integration example we are describing falls under the category of services that *evaluate*. However, a generic Maple service would be able to perform other actions such as *prove* or *decide*. An example of how one might include the directive within the MSDL is given below.

```
<monet:classification>
  <monet:directive-type>
    http://monet.nag.co.uk/monet/directive#evaluate
  </monet:directive-type>
</monet:classification>
```

Again we should expect to be able to prompt the Maple developer to choose from a list of URIs for possible directives. It should be noted that we can include more than one directive.

Algorithm Reference The MSDL schema states that an algorithm description has to refer to the problem it solves and must have a name by which it can be referred to. It can also include taxonomy, complexity and bibliographic information. Figure 4 give an example which includes two bibliographic references and a taxonomic reference.

Algorithmic Properties For this particular example, we may wish to advertise the fact that N-point Gauss-Legendre Integration is exact for polynomials of degree $d \leq 2N - 1$. Algorithmic properties can be listed in MSDL either by using a reference, or using a description using some namespace. For example, the property could be described in English using the xhtml namespace, or perhaps using OpenMath as shown below, which is equivalent to the more human-readable notation:

$$\forall f : f \in \mathbb{R}[x] \wedge degree(f) \leq 2N - 1 \Rightarrow I = \int_a^b f(x)dx$$

```
<monet:implementation>
<monet:algorithmic-properties
  name="Gauss Legendre N-point Quadrature"
  xmlns:om="http://www.openmath.org/OpenMath">
<om:OMOBJ>
  <om:OMBIND>
    <om:OMS cd="quant1" name="forall"/>
    <om:OMBVAR>
      <om:OMV name="f"/>
    </om:OMBVAR>
    <om:OMA>
      <om:OMS cd="logic1" name="implies"/>
      <om:OMA>
        <om:OMS cd="logic1" name="and"/>
        <om:OMA>
          <om:OMS cd="set1" name="in"/>
          <om:OMV name="f"/>
          <om:OMS cd="polysts" name="polynomial_ring"/>
        </om:OMA>
        <om:OMA>
          <om:OMS cd="relation1" name="leq">
          <om:OMA>
            <om:OMS cd="poly" name="degree"/>
            <om:OMV name="f"/>
          </om:OMA>
          <om:OMA>
            <om:OMS cd="arith1" name="minus"/>
            <om:OMA>
              <om:OMS cd="arith1" name="times"/>
              <om:OMI>2</om:OMI>
              <om:OMV name="N"/>
            </om:OMA>
            <om:OMI>1</om:OMI>
          </om:OMA>
        </om:OMA>
```

```
<monet:implementation>
  <monet:algorithm name="Gauss Legendre N-point Quadrature"
                   href="http://www.cs.bath.ac.uk/˜mapma/Monet/num_int_problem_desc.xml">
    <monet:taxonomy taxonomy="http://gams.nist.gov" code="H2a1a1"/>
    <monet:bibliography name="Introduction to Numerical Analysis"
                        href="http://www.ams.org/msnpdf/a0075670.pdf"/>
    <monet:bibliography name="Numerical Recipes:The Art of Scientific Computing"
                        href="http://www.ams.org/msnmain?co3=AND&dr=all&fmt=doc&fn
                              =105&id=98a_65001d&l=100&pg3=TI&pg4=ET&r=1&s3=numeri
                              cal%20recipes%3A%20the%20art%20of%20scientific%20com
                                                                  puting&s4=Books"/>
  </monet:algorithm>
</monet:implementation>
```

Figure 4: Example of an Algorithmic Reference in MSDL

```
      </om:OMA>
      <om:OMA>
        <om:OMS cd="relation1" name="eq"/>
        <om:OMV name="I"/>
        <om:OMA>
          <om:OMS cd="calculus1" name="defint"/>
          <om:OMA>
            <om:OMS cd="interval1" name="interval"/>
            <om:OMV name="a"/>
            <om:OMV name="b"/>
          </om:OMA>
          <om:OMBIND>
            <om:OMS cd="fns1" name="lambda"/>
            <om:OMBVAR>
              <om:OMV name="x"/>
            </om:OMBVAR>
            <om:OMA>
              <om:OMV name="f"/>
              <om:OMV name="x"/>
            </om:OMA>
          </om:OMBIND>
        </om:OMA>
      </om:OMA>
    </om:OMA>
  </om:OMBIND>
</om:OMOBJ>
</monet:algorithmic-properties>
</monet:implementation>
```

3.3 Information Provided by the Service Host

In this section we discuss parts of the MSDL which are added to the Maple developer's input by the Maple wrapper service. These include the following:

- Service Interface Description
- Service Binding Description
- Broker Interface
- Software Details
- Hardware Details
- Supported Semantics
- Actions

We include typical examples of some of these in Figure 5.

3.4 Service Creation and Invocation

In this section we explain how the Maple code supplied by the developer is turned into a Maple service. We describe the process of service invocation by way of a generalised example.

3.4.1 Handling the Maple Code

After the entering all the required parts of the MSDL as described in Section 3.2 along with the Maple code and the interface name which will allow the invocation of the service, the Maple wrapper service creates three permanent files: an MSDL file, a problem description file which is internally referenced within the MSDL file and an XML file containing information about the invocation of the service.

```
<Service name="nameService">
  <Code>
    interfaceName := proc(arg1, ..., argN)
      ...
    end proc:
  </Code>
  <Interface name="interfaceName"/>
</Service>
```

Other parts of the MSDL file are described in Section 3.3.

3.4.2 Calling The Service

The calling of a particular Maple service uses its particular WSDL (see Section 3.3) and a sequence of OpenMath arguments which represent the mathematical inputs. However this WSDL, as we already pointed out in section 3.1, is just a cover for the front-end of our wrapper service and is invoked thus (in Java):

```
String mResult =
  callService(nameService, OMObject arguments[]);
```

The method `callService` performs many operations. Essentially it executes the Maple code for the service called `nameService`, passing it the OpenMath objects from the array `arguments`. The arguments are first translated into strings of valid Maple via the phrasebook and the code is executed by dynamically creating a file (`nameServiceExecutionCode`) containing the following Maple commands:

```
restart:
# Maple code from nameService.xml inserted here
interfaceName := proc(args)
  ...
end proc:
output := MathML[ExportContent]
           (interfaceName(arg1, .., argN)):
fd := fopen("nameServiceResult", WRITE):
fprintf(fd, "%s",output): fclose(fd):
```

where `arg1, .., argN` are the OpenMath input arguments supplied by the client after translation to valid Maple syntax, and `interfaceName` has been lifted from the file `nameService.xml` described above. The Maple code above reads in the procedure definition(s) stored in the file `nameServiceCode` so that it can be executed. This execution code is then run in Maple by running the command line call:

```
String   command = "/opt/maple8/bin/maple "
                   + nameServiceExecutionCode;
Process  p = Runtime.getRuntime().exec(command);
```

WSDL: WSDL information is included in the Service Interface Description of the MSDL. It can either be referenced by a URI as shown below or listed verbatim.

```
<service-interface-description
  name="nameServiceWSDL"
  href="http://agentcities:8090/axis/services/ServiceName?wsdl">
</service-interface-description>
```

Software Details: For our purposes the main software used will be Maple. Thus the MSDL entry should look like:

```
<monet:implementation>
  <monet:software name="maple8"
                 href="http://www.maplesoft.com/products/Maple8/index.shtml"/>
</monet:implementation>
```

Hardware Details: All our services run Maple computations on the Agentcities machine:

```
<monet:implementation>
  <monet:hardware name="Agentcities"
                 href="http://agentcities.cs.bath.ac.uk:8090/axis"
</monet:implementation>
```

Supported Semantics: All services will support OpenMath as the main way to describe mathematical expressions. A list of supported OpenMath symbols could also be given. These would correspond to the symbols supported by the Maple Phrasebook, for example:

```
<monet:semantics name="OpenMath-MathML-CD-Group"
   href="http://monet.nag.co.uk/cocoon/openmath/cdfiles/cdgroups/mathml.html"/>
```

Figure 5: Examples of functional and non-functional constraints

```
Service  serviceMathML2OM = new Service();
Call     callMathML2OM    =
  (Call) serviceMathML2OM.createCall();

String endpoint =
"http://agentcities.cs.bath.ac.uk:8090/axis/'' +
"services/MathMLTranslator";

callMathML2OM.setTargetEndpointAddress(
  new java.net.URL(endpoint));
callMathML2OM.setOperationName(
  new QName(endpoint,"getOpenMath"));

String   ret =
  (String) callMathML2OM.invoke(new Object[] {mResult});

// Get returned an OpenMath object
OMXMLReader tReader = new OMXMLReader(ret);
fOMOutput = tReader.readObject();
return  fOMObject;
```

Figure 6: Java code for making a service call

As can be seen from the Maple execution code, the output is placed in the file `nameServiceResult` and contains a string of Content-MathML. The `callService` method reads this file and returns the string which can then converted to an OpenMath object and returned to the client as in Figure 6.

Finally, the files `nameServiceExecutionCode` and `nameServiceResult` are deleted. Note how the code calls another web service (`MathMLTranslator`) to convert the Content-MathML output from Maple in to OpenMath.

Thus the Maple service exists only in its `nameService.xml` file, and each execution is performed by creating temporary files *on the fly*.

3.4.3 Remarks

WSDL: There is one WSDL file for each Maple service deployed in this fashion. This WSDL file relates indirectly to a single web service though (the Maple wrapper service) which will execute all submitted Maple procedures. This WSDL file will be "dynamic" across all MSDL files for services created using the Maple wrapper service. The details here explain that services are executed by giving as arguments

- A string which is the name of the desired Maple service (`nameService`)
- An array of OpenMath arguments representing the Mathematical arguments being passed to the Maple procedure.

Multiple Service Invocations: The above description works only for single service invocations. In the instance where two clients wish to run the service at the same time, name clash between the temporary files can be avoided by combination with an invocation-unique key.

Some Error Handling Capability: Some new Maple code has been introduced to pick up errors. It handles syntax errors as well as exceptions caused by using, for example, the wrong number of arguments to a Maple procedure. The following code uses Maple's try, catch and finally structure. When a service is invoked, the code shown below is copied into a file to be executed by Maple on the command line. The Java class invoking the service reads the output file to check whether the output is indeed MathML-C. If so, then a translation to OpenMath takes place and the result is returned. Otherwise, it is assumed an error had been encountered and the error string is returned inside an OpenMath error element. This is obviously unsatisfactory, but since only computed results are output in XML-tagged form at present, it is a reasonable compromise which at least ensures the delivery of some information about the service computation.

```
restart:
interface_name := proc(arg1, arg2)
  # some maple code
end proc:
errorflag := 0:
try
  maple_out :=
   interface_name(parse(arg1, statement),
                  parse(arg2, statement)):
catch:
  errorflag := 1:
  output := lastexception[2]:
finally:
  if (errorflag = 0) then
```

```
    output := MathML[ExportContent](maple_out):
  end if;
  fd := fopen("/tmp/serviceNameOutput", WRITE):
  fprintf(fd, "%s",output):
  fclose(fd):
end try:
```

Handling Multiple Results: We have yet to properly address the case of multiple results. At present the Maple code written typically returns a set or list when the service produces multiple results. For example, the numerical integration routine returns a list comprising the approximate solution together with a bound on the accuracy of that solution. This is translated to an OpenMath list object. However the problem description states that the service returns two separate objects. It is not yet clear how this should be handled. One approach we considered was to check the length of the list (in the Maple code above) and return each element of the list in a separate numbered file. Then the Java code reads the problem description to find the number of required outputs and looks for that many files. However this relies on using the `nops()` function in Maple. We found problems with such Maple types as fractions, where `nops()` returns two, where the operands are the numerator and the denominator.

Service Response Object: With a view to a forthcoming explanation ontology [4] responsible for providing a meta-structure in which to state the nature and existence of explanations, we decided to package the results of service invocations inside a service response object. This Service Response object carries some extra information such as the service name, input arguments, problem description, Maple procedure, code executed by Maple and the Maple return in both syntaxes – Maple and OpenMath. It also contains a log of all the commands executed by the service invocation in order to ease the task of testing any possible problem during the execution of such maple code.

Argument Types and Bindings: Our service could include the capability to check the client's input argument types against input types in the problem description. This would typically involve calling another service to reason mathematically about the types of the OpenMath objects. This will undoubtedly be done by part of the broker, when attempting to match queries to services. However, it would be useful for our service to check the arguments that are actually finally supplied by the client, so as to avoid having to handle Maple errors.

3.5 Demonstrations

The Maple Wrapper service URL is `http://agentcities.cs.bath.ac.uk:8090/axis/SMaple/services/index.jsp`, whence one may add, remove, update, list and invoke Maple services. When adding a Maple service the client is prompted to submit details such as service name and Maple code. The client must also specify the number of inputs, outputs, pre-conditions and post-conditions, and is then prompted to enter descriptions of these in OpenMath. When invoking a Maple service through the Maple Wrapper service, a client fills out fields providing the service name and input arguments in OpenMath.

The demonstration provided here is still quite basic, and does not yet allow for complete MSDL descriptions of the Maple services. The WSDL for the Maple wrapper service can also be viewed at `http://agentcities.cs.bath.ac.uk:8090/axis/services/MapleWrapperService?wsdl`.

4. EXAMPLE MAPLE SERVICES

We now describe two Maple services which have been deployed using the Maple wrapper service described above.

4.1 A Symbolic Integration Service

The first example is a simple service which uses Maple's `int` command to compute a definite integral symbolically. The Maple code for this service is detailed in the appendix to [6], while a valid Maple execution of this service is:

```
symbolic_def_integration(((x)^(2)/2),x,-1,1);
```

4.1.1 A Problem Description

A suitable problem description for our symbolic integration service is given below. It details as inputs a function (the integrand), the upper and lower limits of integration, and the variable with which to integrate with respect to. A snippet of this problem description from the MSDL document is given in the appendix to [6]. Extensions to the service could be to include checks on the suitability of the integrand and the integration limits. Thus we would include pre-conditions on the input, such as continuity of the integrand, to the problem description.

Input:	1. $F : \mathbb{R} \to \mathbb{R}$ 2. x 3. $a \in \mathbb{R}$ 4. $b \in \mathbb{R}$
Output:	1. $I \in \mathbb{R}$
Post-Condition:	1. $I = \int_a^b f(x)dx$

4.1.2 Sample Invocation of the Service

Invoking this Maple service with the arguments described above (and in OpenMath in the appendix of[6]) results in the following correct response in OpenMath.

```
<OMOBJ>
   <OMA>
      <OMS cd="nums1" name="rational"/>
      <OMI>1
      </OMI>
      <OMI>3
      </OMI>
   </OMA>
</OMOBJ>
```

Also included in the appendix of [6] are extracts from the actual SOAP messages sent to and received from the service upon invocation. These messages show how the OpenMath input arguments and the answer are packaged within the call to the web service.

4.2 A Root-Finding Service

This section describes the Maple code behind a simple root-finding service which uses Maple's `solve` function. The Maple procedure accepts a set of equations, and a set of dependent variables to solve for. The procedure returns a set of solution sets. The Maple code for this service is listed in the appendix of [6]. This can be invoked in Maple by (for example):

```
simple_solver({x-y-1, x^2*y^2}}, {x,y});
```

4.2.1 A Problem Description

This problem description is less straightforward. For a simple root finding service which uses Maple's `solve` command, the inputs

Input:	1. A set of m polynomial equations in n unknowns: $G = \{g_1, \dots g_m\}$ 2. A set of n variables: $\{x_1, x_2, \dots, x_n\}$
Output:	1. A set of l solutions: $\{(a_{1,1}, a_{2,1}, \dots a_{n,1}), (a_{1,2}, a_{2,2}, \dots a_{n,2}), \dots (a_{1,l}, a_{2,l}, \dots a_{n,l})\}$, where $a_{1,j}, a_{2,k}, \dots a_{n,j} \in \mathbb{C}$
Post-Condition:	1. The l solutions satisfy the m equations: $g_i(a_{1,j}, a_{2,j}, \dots, a_{n,j}) = 0, \quad \forall i \in 1, \dots, m, j \in 1, \dots, l$

Figure 7: A root finding service description

should be a set of equations, and a set of variables to solve with respect to. The output should be a set of solutions, with the post-condition being that the solutions satisfy the original equations. The description appears in Figure 7.

4.2.2 Sample Invocation of the Service

If we try this service on the set of equations $G = \{x-y-1, x^2y^2\}$, and the variables $\{x, y\}$ (expressed in OpenMath in the appendix to [6]), the result is $\{\{y = 0, x = 1\}, \{y = -1, x = 0\}\}$, expressed in OpenMath as:

```
<OMOBJ>
   <OMA>
      <OMS cd="set1" name="set"/>
      <OMA>
         <OMS cd="set1" name="set"/>
         <OMA>
            <OMS cd="relation1" name="eq"/>
            <OMV name="y"/>
            <OMI>0
            </OMI>
         </OMA>
         <OMA>
            <OMS cd="relation1" name="eq"/>
            <OMV name="x"/>
            <OMI>1
            </OMI>
         </OMA>
      </OMA>
      <OMA>
         <OMS cd="set1" name="set"/>
         <OMA>
            <OMS cd="relation1" name="eq"/>
            <OMV name="y"/>
            <OMI>-1
            </OMI>
         </OMA>
         <OMA>
            <OMS cd="relation1" name="eq"/>
            <OMV name="x"/>
            <OMI>0
            </OMI>
         </OMA>
      </OMA>
   </OMA>
</OMOBJ>
```

Again the SOAP packets describing the request and response appear in the appendix to [6].

5. CONCLUSIONS AND FUTURE WORK

There are several points of discussion as to the merits of the web service implemented. The Maple wrapper service assumes absolutely no knowledge of web services or MSDL. Furthermore, we just expect the Maple developer to know OpenMath in order to complete a problem descriptions or describe algorithmic properties. Each mathematical-service description (mostly contained in the MSDL file) is generated by the wrapper service. All services are hosted on the same server as the wrapper service itself.

We have designed our service so that it is not necessary to restart Axis when new services are added, or existing services updated. So that others may experiment, we have constructed a down-loadable package containing all resources needed (web pages,web services,etc). The installation of such package allows the user to deploy services in any machine via JSP pages and web services, as we generate a separate WSDL for each service.

Directions for future work include describing a wider range of mathematical services to establish the adequacy of MSDL, building interfaces to other computer algebra systems, tackling the significant problem of sessions, retention of data (this might be large in some cases, and it might be pointless to ship to-and-fro between client and server) and provenance of data. However, these latter issues are generic and are of concern to the wider e-Science community.

6. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[2] S. Buswell, O. Caprotti, and M. Dewar. Mathematical Service Description Language: Initial Draft. Technical report, Technical Report Deliverable, The MONET Consortium, March 2003. Available from `http://monet.nag.co.uk`.

[3] S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 implementation. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 241–248. ACM Press, 1997.

[4] J. Davenport. Mathematical Explanation Ontology: Draft. Technical report, Technical Report Deliverable, The MONET Consortium, March 2003. Available from `http://monet.nag.co.uk`.

[5] F. Hildebrand. *Introduction to Numerical Analysis*. McGraw Hill, 1956.

[6] marc Laurent Aird. Symbolic Service Initial Beta Version. Technical report, Technical Report Deliverable, The MONET Consortium, July 2003. Available from `http://monet.nag.co.uk`.

[7] National Institute for Standards. GAMS Guide to Available Mathematical Software. `http://gams.nist.gov/`, February 2003.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1987.

[9] Technical University of Eindhoven. RIACA: OpenMath Library. `http://www.riaca.win.tue.nl/products/openmath/lib/index.html`, February 2003.

[10] The Apache Project. Jakarta Home Page. `http://jakarta.apache.org`. Last accessed January 2004.

[11] The MONET Consortium. MONET Home Page. `http://monet.nag.co.uk`.

[12] The OpenMath Society. OpenMath website. `http://www.openmath.org`, February 2003.

[13] S. Turner. Numerical Service Initial Beta Version. Technical report, Technical Report Deliverable (D10), The MONET Consortium, March 2003. Available from `http://monet.nag.co.uk`.