An Architecture for Distributed Mathematical Web Services

Elena S. Smirnova, Clare M. So, Stephen M. Watt

Ontario Research Centre for Computer Algebra (ORCCA)
Department of Computer Science
University of Western Ontario
London, Ontario, CANADA. N6A 5B7
{elena,clare,watt}@orcca.on.ca

Abstract. This paper describes technologies to create and maintain a problem solving environment based on a framework for distributed mathematical web services. Our approach allows clients to access mathematical computational facilities through a uniform set of network protocols, independent of the software packages that provide the end functionality. The author of a mathematical web service need know only the specifics of the system in which the mathematical computations are performed, e.g. Maple, Mathematica or Fortran with NAG library. Whatever additional network service code that is necessary (e.g. Java, WSDL, etc), is generated and configured automatically. This paper presents a brief architectural overview of the entire framework, and then gives a detailed description of the design and implementation of the tools for mathematical servers for this environment. A set of mathematical web services currently deployed are described as examples.

1 Introduction

Over the past decades mathematical software systems have become increasingly powerful and complex. Each major system has its own strengths and areas in which it surpasses the others, often provided by complex packages or libraries. This picture changes constantly as new versions of systems are released. Considerable expertise required to know which package is the best for which problem at any given time. Users are therefore often unaware of existing software solutions to their mathematical problems. Even if they are aware they may not be willing to maintain up-to-date licenses for multiple software systems only occasionally used. We see this situation becoming more challenging over time, and its solution is the object of the MONET project.

MONET is a European ESPRIT project for *M*athematics *O*n the *NET*, with the goal of designing a framework for the provision and brokerage of mathematical web services. The partners in this consortium are the Numerical Algorithms Group Ltd (NAG), the University of Bath, Stilo Technology Ltd, the University of Manchester, the Technical University of Eindhoven, the University of Nice, and the Ontario Research Centre for Computer Algebra at the University of

Western Ontario. The architecture devised by the MONET consortium involves (1) a number of software components, including brokers, planners and servers, (2) various ontologies representing mathematical problem domains and services, and (3) specific languages to communicate mathematical problems, results and explanations. An overview of this architecture is given in Section 2, and a detailed description is given in the report [1]. While the best approach for service brokers and planners remains the subject of on-going research, the strategy for providing the specific mathematical services can be declared complete. This is the main subject of the present paper.

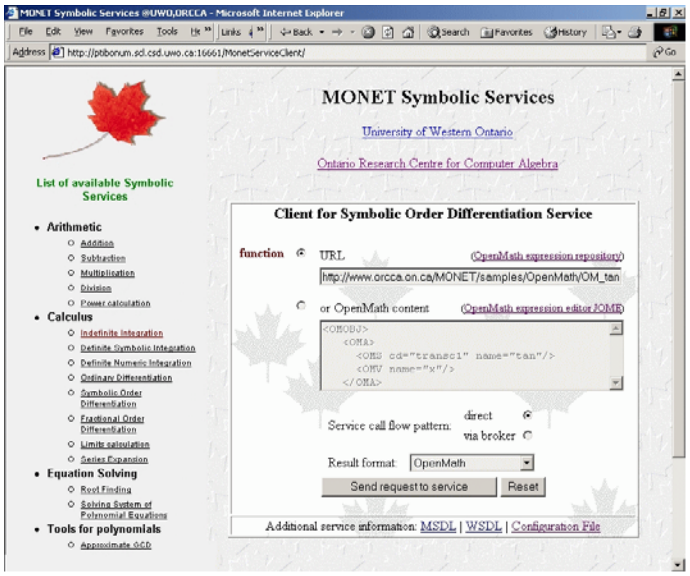
We have developed an approach whereby (1) the author of a mathematical web service need know only the software system he or she uses to provide the service, and (2) the user of the mathematical service need not be aware of the implementation language. For example, an expert in symbolic computation, proficient in Maple, can develop a set of Maple packages solving a specific problem. This expert can then use our framework to create a mathematical web service to solve this problem. Without additional effort on the part of the service author, the service will accept requests in a system-neutral protocol, based on OpenMath [9][10]. We describe this architecture for mathematical web services in Section 3. A number of issues arising in the implementation are described in Sections 4, 5 and 6.

We believe that both services and clients benefit from this architecture. Clients do not have to have the necessary mathematical software installed locally, and are not required to know the syntax and calling sequences of multiple mathematical software packages. Service providers can expose their most recent mathematical software to a larger target audience, reduce maintenance costs for old versions, and potentially derive transaction-based revenue. Once a sufficient number of services are deployed, the MONET framework could be applied to a wide spectrum of applications, including professional technical services, mathematical education and distance learning, and research support in a number of domains. As mathematical systems become ever more complex, this approach to service discovery and planning could conceivably be used within individual systems.

In order to place our work in context in the world of mathematics-related web services, we refer to some other projects in this area, including *MBase* [25] and *H-Bugs* [11]. MBase is a extensive database for mathematical theory, symbol, definition, axiom and theorem. It contains a variety of mathematical knowledge description, but does not particularly aim to offer any implementation of algorithms or methods for the solution of mathematical problems. H-Bugs is an investigation to distributed system based on MONET architecture, as demonstrated, for theorem proving. The main topic of this paper is the description of the architecture for providing *mathematical computation* web services within the MONET framework.

As a first example, we have developed and deployed the following symbolic mathematical web services at University of Western Ontario [1] (see Fig. 1): Arith-

[1] URL: `http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient/`


MONET Symbolic Services @UWO,ORCCA - Microsoft Internet Explorer
File Edit View Favorites Tools He Links Back Search Favorites History
Address http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient/
Go

MONET Symbolic Services

University of Western Ontario

Ontario Research Centre for Computer Algebra

List of available Symbolic Services

- Arithmetic
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Power calculation
- Calculus
  - Indefinite Integration
  - Definite Symbolic Integration
  - Definite Numeric Integration
  - Ordinary Differentiation
  - Symbolic Order Differentiation
  - Fractional Order Differentiation
  - Limits calculation
  - Series Expansion
- Equation Solving
  - Root Finding
  - Solving System of Polynomial Equations
- Tools for polynomials
  - Approximate GCD

Client for Symbolic Order Differentiation Service

function [✓ URL] (OpenMath expression repository)

http://www.orcca.on.ca/MONET/samples/OpenMath/OM_tan

or OpenMath content (OpenMath expression editor JOME)

```
<OMOBJ>
  <OMA>
    <OMS cd="transc1" name="tan"/>
    <OMV name="x"/>
  </OMA>
```

Service call flow pattern: [✓ direct] via broker

Result format: OpenMath

Send request to service | Reset

Additional service information: MSDL | WSDL | Configuration File

Fig. 1. MONET web services page at University of Western Ontario

metic expression simplification, Indefinite integration, Definite symbolic integration, Definite numeric Integration, Ordinary differentiation, Fractional-order differentiation, Symbolic-order differentiation of rational functions, Limit calculation, Series expansion, Approximate GCD, Root-finding service (including parametric solutions) and Solving systems of polynomials.

2 General MONET Architecture

2.1 Main Components

There are three components in MONET architecture: client, broker and server (see Fig. 2). Servers expose their services by registering with brokers. Clients send queries to a broker asking if web services for solving specific kinds of mathematical problems are available. Brokers resolve clients' queries by looking for suitable mathematical services available and passing the clients' requests to them. After this, services try to solve the given mathematical problems using the capacities of their mathematical solving engines. Finally, clients obtain the result of the computation and possibly a meaningful explanation.

2.2 Mathematical Broker

The Mathematical Broker presented in [5] is a special component in this architecture. It is assumed to be a sophisticated search engine and planner for

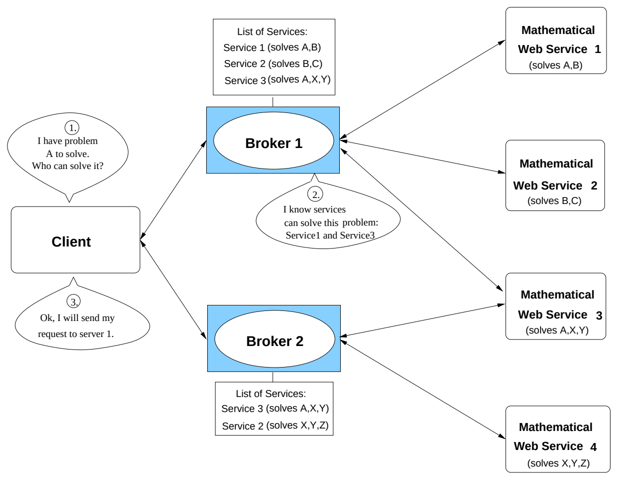

List of Services:
Service 1 (solves A,B)
Service 2 (solves B,C)
Service 3 (solves A,X,Y)
Mathematical Web Service 1 (solves A,B)
1. I have problem A to solve. Who can solve it?
Broker 1
Mathematical Web Service 2 (solves B,C)
Client
2. I know services can solve this problem: Service1 and Service3
Mathematical Web Service 3 (solves A,X,Y)
3. Ok, I will send my request to server 1.
Broker 2
List of Services:
Service 3 (solves A,X,Y)
Service 2 (solves X,Y,Z)
Mathematical Web Service 4 (solves X,Y,Z)

Fig. 2. The general scheme of the MONET Architecture

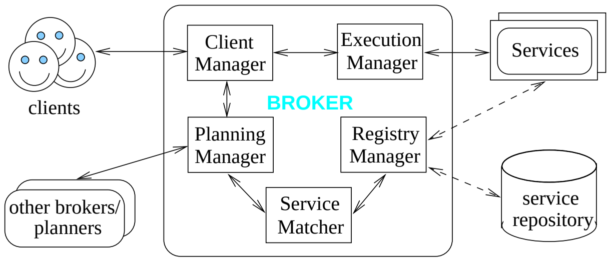


Fig. 3. The general scheme of the MONET Broker

complex mathematical problems. Its goals are to match the characteristics of given mathematical problem to web services available and, if no single service can solve the given problem by itself, to divide the problem into smaller portions that can be solved by available services. In general, matching requests for arbitrary mathematical problem is intractable. However, we seek to provide a solution for a wide range of problems that appear in practice.

2.3 Languages and Standards Used

Languages and standards have been developed by the OpenMath society and the MONET consortium to describe mathematical problems and communicate the results in an unambiguous way. OpenMath[9][10] is an extensible standard to encode the semantics of mathematics in a system-neutral format. It was chosen for communicating mathematics in this architecture because, unlike Content MathML, it can be extended natively to cover new areas of mathematics by writing new Content Dictionaries. The Mathematical Service Description Language

(MSDL)[4] is a specialized format that describes mathematical web services in the MONET architecture. It is needed when servers register the services with brokers. There are others languages developed by MONET consortium, but these two are used by servers to expose and provide services.

3 Architecture of Mathematical Web Services

In the MONET architecture mathematical web services are provided without revealing their implementation structure. A mathematical web service can be provided by a stand-alone C program, or by an interface to a package from a system such as Axiom[22], Derive[23], Maple[20], Mathematica[21] or Matlab[24]. This approach aims to be flexible and extensible, to allow easy switching between mathematical systems, and to allow the easy creatoin of new services.

In this section we describe the architecture we have designed for the construction of such a Mathematical Web Server. In our description we show the service being provided by a Maple program, but in principle any other package could be used.

3.1 Mathematical Web Server Environment

The first design decision for the Mathematical Server is that all services deployed on it to run independently, each reachable at its own unique URL and offering its own functionality. All of these services are enclosed within a specially designed software package, sometimes called the *wrapper tool*, and may be maintained by several managers. Together, these form the *Mathematical Web Server Environment*.

One of the essential components to the Mathematical Web Server is its *mathematical engine*, usually provided by a a package within some computer algebra system.

The general scheme of organization for the Mathematical Server is shown in Figure 4. It shows that each mathematical service is assigned to several components, which all together are responsible to fulfill service's functions.

The principal information about each service is provided by the *service configuration file* that contains tree parts: (1) a formal service description, using MSDL [4], (2) a service interface to a mathematical software system, and (3) an actual service implementation provided as code for that system. It is possible for alternative implementations to be given, in which case parts (2) and (3) have multiple components.

The wrapper tool is a container for a generic mathematical service, and its specific behaviour is determined by the service configuration file.

Outside the mathematical engine, there are two core managers to maintain the network interface to this mathematical server environment: one is responsible for new service creation and installation, and the other for invocation of a service installed.

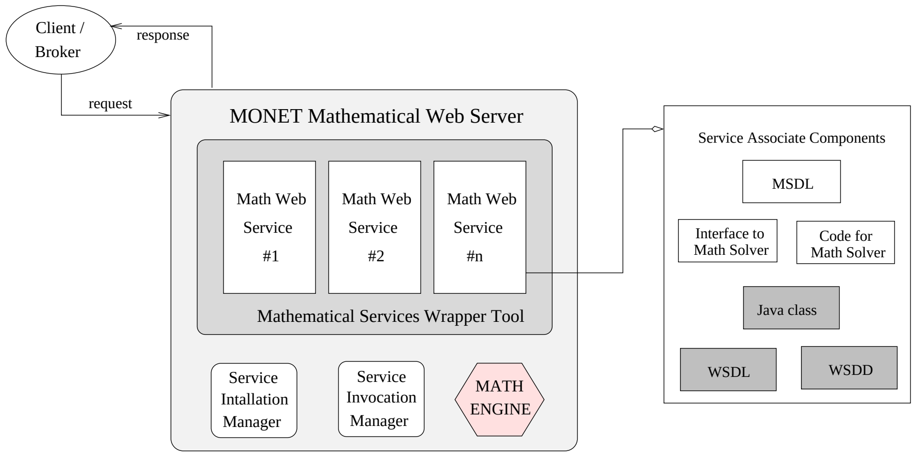


Fig. 4. The general scheme of MONET Mathematical Server Architecture

3.2 Mathematical Service Organization

Mathematical Services are organized in such a way that the author of a new service need not know about web servers and services, Java or the various XML technologies in order to create a new service. Instead, the service's authors provide code written for a mathematical system that implements their service. The author must separately indicate the name of the main function that enters this implementation.

As shown in the Figure 4, each service is associated with three original pieces of information and three generated components. The original information about the service includes the formal service description given in MSDL, service interface to the mathematical engine of the server and code for the service implementation to be executed by mathematical engine. Generated associates of the service are service core Java class, web service deployment descriptor (WSDD) [8] and web service description language file (WSDL) [7].

Service Configuration File The original information about a mathematical service is provided by the service author and stored in an XML-based configuration file. This file is the item in the mathematical service infrastructure that uniquely and completely represents the service implementation.

The service description file consists of three parts: the service MSDL skeleton (to be completed automatically during service installation), service interface to mathematical engine (for example Maple or Axiom), and service implementation (code written in the language of the mathematical system used).

The structure of the configuration file has the pattern indicated below.: Note that a file may contain descriptions for more than one service, especially if it makes sense when several services share parts of an implementation.

```
<mathServer>
    <msdl>
        <service name="sevice_A">
           {MSDL skeleton for service A}
        </service>
        {MSDLs for other services}
    </msdl>

    <services>
        <service name="service_A" call="function_call_for_service_A"/>
        {interface for other services}
    </services>

    <implementation language = "math_solver_name">
        {implementation for each service using
         the languageof the corresponding solver}
    </implementation>
</mathSever>
```

The configuration file for each service should be available for the wrapper tool at any time after the service is installed on the Mathematical Server.

Generated Service Associates The configuration file statically describes service functionalities. In order to fulfill them dynamically we still need additional descriptors and implementation tools. Those components are service implementation Java class, WSSD and WSDL files.

All of them are automatically created by the Mathematical Solver Environment installation engine: a Java class is created according to the information provided in the service configuration file, then both WSDD and WSDL descriptors are derived from this Java class API.

These generated components are then used to deploy, expose and run the service. During service installation, the created Java class is placed in a proper location in the web server file system, so it is ready to maintain service calls. WSDD is used to deploy service on the web server, and WSDL is exposed to the outside world to provide MONET brokers and clients with the information about service interface (see Fig. 5).

Realization of Service Functions The main functions of each mathematical web service are carried out by a combination of a service core Java class, the server Invocation Manager and information from the service configuration file.

The main idea of the service organization is that its Java class receives all requests from the outside (client or brokers), extracts information about mathematical arguments (if any) passed with the request and then calls service invocation engine. The Invocation Manager in turn creates a program for the solving engine in real time, using information from service configuration file and the mathematical arguments from the service call. This program is passed to the server's mathematical engine for execution. The result is retrieved by the same Invocation Manager, wrapped into a service response message and sent back to the client or broker (see Fig. 6).

4 Details of the Architecture Implementation

4.1 Technologies Used

The list of technologies involved in the process of developing Mathematical Web Server includes several network protocols: SOAP [6], TCP, HTTP, etc., various products from Apache software foundation: Jakarta Tomcat [16], Axis [17] and Ant [18], different computer algebra systems such as Maple [20] and Axiom[22], as well as Java SE [14] from Sun Microsystems.

4.2 Core Components of the Mathematical Web Server Architecture

As mentioned in section 3, the implementation of the Mathematical Web Server includes installation and invocation managers. The installation manager takes care of new service creation and maintaining (installation/deinstallation). The invocation manager is an engine that handles calls to services from the outside. There is also a third component of Mathematical Web Server; it is responsible for installation of mathematical server itself.

4.3 Mathematical Web Server Installation

The Mathematical Web Server installation process sets up a web server in order to prepare it for running mathematical web services.

Assuming Apache Tomcat and Apache Axis are already installed on a web server, the procedure of setting it up for MONET Web services is performed by running a script from the installation kit.

4.4 Mathematical Web Service Installation

The Mathematical Service Installation Manager is implemented as a combination of Java programs and shell scripts. Each of the Java programs is responsible for carrying out one or more steps of service installation process. The whole instillation is driven by a manager script. As described in the section 3.2, all necessary information about a service is provided in its configuration file. In order to install a service, the Installation Manager needs to have access to this file. To register a new service with the MONET Broker, the Installation Manager also requires the URL of the Broker.

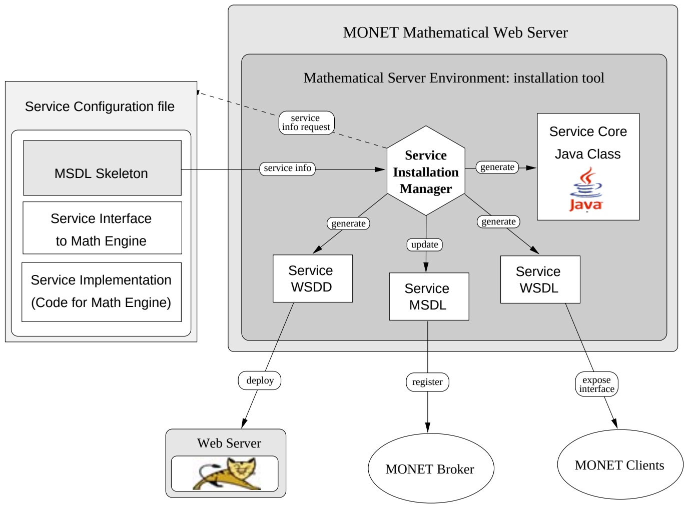

MONET Mathematical Web Server
Mathematical Server Environment: installation tool
Service Configuration file
MSDL Skeleton
Service Interface to Math Engine
Service Implementation (Code for Math Engine)
service info request
service info
Service Installation Manager
generate
Service Core Java Class
Java
generate
update
generate
Service WSDD
Service MSDL
Service WSDL
deploy
register
expose interface
Web Server
MONET Broker
MONET Clients

Fig. 5. MONET Mathematical Service Installation

While installing a new service(see Fig. 5), the Installation Manager

- parses configuration file to extract the information about service interface,
- creates Java class that will implement the service on the web server,
- generates server deployment descriptor (WSDD),
- deploys the service on the web server,
- retrieves WSDL (descriptor of web service interface to clients) created by the server,
- according to the current service installation, updates its MSDL skeleton to a fully functional mathematical service descriptor.

After installation is complete the service can be seen from outside by MONET clients and brokers, and it is ready to handle requests from them.

4.5 Mathematical Service Invocation

When a mathematical service receives a request with a mathematical query, the service core Java class retrieves the input arguments from this query and calls the service Invocation Manager.

The Mathematical Service Invocation Manager is implemented as a combination of several Java libraries and auxiliary packages for mathematical engine systems, designed to fulfill service functions according to the information provided in the service configuration files.

The following steps (also shown as scheme in the Fig. 6) are executed every time a mathematical service is invoked:

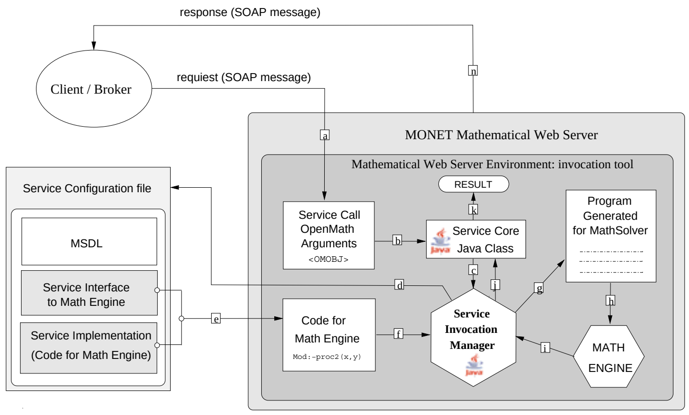

response (SOAP message)
Client / Broker
requiest (SOAP message)
MONET Mathematical Web Server
Mathematical Web Server Environment: invocation tool
Service Configuration file
MSDL
Service Interface to Math Engine
Service Implementation (Code for Math Engine)
Service Call OpenMath Arguments <OMOBJ>
RESULT
Service Core Java Class
Program Generated for MathSolver
Code for Math Engine Mod:-proc2(x,y)
Service Invocation Manager
MATH ENGINE
a
b
c
d
e
f
g
h
i
j
k
n

Fig. 6. MONET Mathematical Service Invocation

1. The service Java class calls the Invocation Manager with the following parameters
   – the reference to the service configuration file
   – an array of mathematical arguments from the query to the service
   – optional mathematical formats, if client set preference for encoding of service input and output (see section 5).
2. The Invocation Manager parses the configuration file and extracts the service implementation and service interface.
3. The service arguments, by default encoded in OpenMath, get parsed and prepared for the conversion to the internal syntax of the mathematical engine.
4. The Invocation Manager *generates* the program to be executed by the mathematical solving engine. This is based on the implementation part from the service configuration file, the service interface to this system and the mathematical arguments from service request.
5. The generated code is passed to the mathematical engine for execution.
6. The result from mathematical engine returns to the Invocation Manager. There it gets converted to an appropriate mathematical format and encoding (see section 5.2), afterward it is passed back to service core Java class.
7. The service Java class wraps the answer into a message using the SOAP object exchange protocol and sends it back to the MONET client.

5 Mathematical Web Service Input and Output

5.1 Input Format

All MONET mathematical services accept input arguments as OpenMath expressions. Each expression can be encoded as plain string, an XML DOM object[13] or a RIACA OpenMath object[12]. The reason to support all of these encodings was to experiment with a choice of possible interfaces to clients and brokers while using the same service.

A string is the simplest way to format a request. This does not require any extra libraries, nor serialization tools on a client side. However, in case of string-based inputs there is a hight probability of submitting invalid OpenMath within service requests.

An XML Document Object Model (DOM) is used to represent parsed XML documents in as a tree structure. It is a moderately more complex encoding of OpenMath objects that at least can ensure that the passed arguments represent are valid XML.

The RIACA format is specially designed to represent OpenMath objects. It guaranties that inputs sent are valid OpenMath, but in this the case client has to know about the RIACA library, have it installed locally and perform the serialization of RIACA objects when sending SOAP messages with service queries.

5.2 Output Format

The mathematical services currently implemented return string object encoding OpenMath expression, containing the list of results or error message.

MEL format If requested by a client, the answer from a service can be wrapped in a service response object, using the MONET Mathematical Explanation Language (MEL) [2] ontology. We do not offer this option by default, since when the result from a service is sent back to the MONET Broker, the later will provide the wrapping, possibly including supplementary information, such as explanation from a Planning Manager or Execution, etc.

Additional mathematical formats for service output In addition, our current implementation of MONET mathematical services supports several other mathematical formats to encode service results. Users may set their preferences while calling a service, and the result can be given in any of the following formats: OpenMath, Content MathML, Presentation MathML and LaTeX. This is an experimental facility that can be used in environments that integrate several formats into represent mathematical objects.

5.3 Conversion Between OpenMath and Math Engine Language

Conversion between OpenMath and language of the chosen solvers is important in offering mathematical web services. All mathematical expressions must be encoded in OpenMath, a system-neutral format, but not all solvers can accept input and return output in OpenMath.

Conversion between OpenMath and strongly-typed languages, such as Axiom, is not difficult because type information is apparent in both formats. Conversion between OpenMath and dynamically typed languages, such as Maple, does not have a trivial solution because the semantic information for expressoins in these languages is often implicit. It is not practical to force all service authors to use only solvers that use strongly-typed languages, because solvers that use dynamically-typed expressions can be very useful in practice. We use Maple to demonstrate the challenges and our solution in converting between OpenMath and a dynamically typed language. Similar issues would be encountered when service authors try to offer mathematical web services using another mathematical engine having a dynamically-typed language.

The conversion between OpenMath and Maple is described using a collection of *Mapping files*, typically one for each OpenMath CD in use. Mapping files describe the correspondence between OpenMath and Maple, and a single mapping file is used to drive both directions of the conversion. An entry in a mapping file contains a fragment of OpenMath markup and the corresponding structure of Maple command. When designing the mapping files, some differences between OpenMath and Maple are generalized. Order of arguments and corresponding Maple command are taken into account. If the mapping file's design cannot be used to describe the conversion, a Maple procedure can be attached instead.

Converting from OpenMath to Maple is mostly straight-forward. The correspondence can usually be determined by the OpenMath symbol (`<OMS>`) alone. Difficulties in this direction of conversion still exists because of the differences in the design of the languages or in the form of certain expressions.. For example, the partial differentiation operation in Maple takes a list of bound variables to indicate with respect to which the differentiation is to be done. The corresponding OpenMath function takes the indexes of bound variables in a list to achieve the same purpose. In the worst case, an OpenMath expression may not have a Maple correspondence at all. In this case, the converter throws an error notifying the users that certain OpenMath symbols are not handled by this converter.

Maple to OpenMath conversion is less straight-forward because a Maple expression is subject to ambiguities. Ambiguities exist in a Maple expression because Maple is a dynamically-typed language and the meaning of an expression cannot be determined by the symbolic function compositions alone. As a simple example, Maple uses the same function call ("`diff`") for both univariate and partial differentiation. OpenMath has two distinct symbols (`<OMS>`) for these mathematical concepts. Some ambiguities can be resolved by inspecting the number of types of arguments. For ambiguities that cannot be resolved in this manner, we develop and employ policies to specify what to do with such cases. The first policy is to resolve the ambiguity by an extra Maple procedure supplied when the converter is invoked. An example strategy in this policy is to choose first possible OpenMath correspondence found. Second policy is to generate a pseudo OpenMath representation that can be dealt with later. In the worst case, the third policy is used to throw an error notifying the user that such ambiguity cannot be resolved.

6 Using Other Mathematical Engines with the Server

As mentioned in section 3.2, different math engines can be used with this service architecture. For the author of the service, switching between mathematical engines means only editing the service configuration file. For the developer of a Mathematical Server, changing its mathematical engine requires replacing components of the Invocation Manager that are specific to the mathematical software used, however the second part of the Mathematical Server – Installation Manager remains the same.

6.1 Changes to service configuration file

The configuration file allows an author to specify the name of the mathematical engine, and give an implementation for the engine with that name. It also permits to have more than one implementation for the same service, using alternative mathematical engines. For example the service for definite integration may use system Derive, Axiom or Mathematica instead of Maple, or it may use all four of them. In this case all that is required from the author of the service is to change the implementation part of service configuration file. If one decides to use all of the above computer algebra systems to implement definite integration service, its configuration file might be re-written as the following:

```
<mathServer>
 <msdl>
  { MSDL for Definite Integration Service }
 </msdl>

 <services>
  <service name="DefiniteIntegrationService" call="monetDefInt"/>
 </services>

 <implementation language = "maple">
   monetDefInt:=proc(function,var,lower_limit,upper_limit);
          int(function,var=lower_limit..upper_limit);
   end:
 </implementation>
 <implementation language = "derive">
   monetDefInt(f,x,a,b,e) := PROG( e:=INT(f,x,a,b), return(e))
 </implementation>
 <implementation language = "mathematica">
   monetDefInt[f_,x_,a_,b_] := Integrate[f,{x,a,b}]
 </implementation>
 <implementation language="axiom">
)clear completely
monetDefInt(f,x,a,b)==
  integrate(f,x=a..b)
 </implementation>
</mathServer>
```

6.2 Changes to the Mathematical Server components

In order to enable capabilities of Mathematical Server to handle services using various mathematical engines, the developer of a particular instance of the mathematical server has to replace the elements of the Invocation Manager that depend on the mathematical engine. These are the code generation, the tools for conversion between OpenMath and the language of this system and an adapter that allows to plug the system software into Mathematical Server environment.

Essentially, this means providing new code for Java classes implementing these functions. The good news is that the stubs of these classes, once created, can be reused by simply pointing to the location of their files during Mathematical Server installation. Currently we have created such code stubs for two computer algebra systems: Maple and Axiom. None of this is completely straightforward, but fortunately these tasks need be done only once for each system.

The whole structure of Mathematical Server, as well as its tools assigned to install and maintain new services remain the same, since they do not depend on the mathematical engine.

6.3 Current Status and Future Work

We have successfully applied this technique to switch between two computer algebra systems, Maple and Axiom. We have not yet experimented with substituting the mathematical engine with a Fortran program using NAG libraries, but we believe that the organization of the web server and mathematical web services will easily allow this.

7 Conclusion

We have described an approach to providing mathematical web services in which authors wishing to provide services need know only the mathematical software packages with which they are already familiar. No knowledge is required of Java, WSDL or other web technologies. We see this as a significant benefit over previous approaches, given the rarity of individuals with expertise in both web technologies and mathematical software.

For each mathmatical software system used to implement services, a translator program to and from OpenMath must be provided. This is required only once for each system, however.

The client-side interface to the mathematical services is suitable for use both by software systems and end users. It is currently used to serve a test web site providing a selection of symbolic mathematical services at `http://ptibonum.scl.csd.uwo.ca:16661/MonetServiceClient`. It is also used for on-going experimentation with complex service brokers that plan solution strategies combining services from different mathematical software systems.

References

[1] Stephen Buswell,Olga Caprotti and Mike Dewar, MONET Architecture Overview, Deliverable 4, The MONET Consortium (2002)

[2] Yannis Chicha, James Davenport, David Roberts, *Mathematical Explanation Ontology*, Deliverable 7, The MONET Consortium (2004)

[3] Olga Caprotti, David Carlisle, Arjeh M. Cohen, Mike Dewar, Mathematical Problem Description Ontology, Deliverable 11, The MONET Consortium (2003)

[4] Stephen Buswell, Olga Caprotti, Mike Dewar, *Mathematical Service Description Language* , Deliverable 14, The MONET Consortium (2003)

[5] Walter Barbera-Medina, Stephen Buswell, Yannis Chicha, Marc Gaetano, Julian Padget, Manfred Riem, Daniele Turi, Broker API, Deliverable 18, The MONET Consortium (2003)

[6] W3C, Simple Object Access Protocol (SOAP) 1.1, `http://www.w3.org/TR/SOAP`, (2003-2004).

[7] W3C, Web Services Description Language (WSDL) 1.1, `http://www.w3.org/TR/wsdl.html`

[8] W3C, Web Service Deployment Descriptor (WSDD) 1.1, `developer.apple.com/documentation/WebObjects/Web_Services/Web_Services/chapter_4_section_7.html`

[9] OpenMath, `http://www.openmath.org/cocoon/openmath`, (1999-2004)

[10] An OpenMath v1.0 Implementation, S. Dalmas, M. Gaetano and S. M. Watt, Proc International Symposium on Symbolic and Algebraic Computation (ISSAC 1997), pp 217-224, ACM Press 1997.

[11] Brokers and Web-Services for Automatic Deduction: a Case Study, C. S. Coen and S. Zacchiroli, Proc Calculemus 2003

[12] RIACA OpenMath Library, `http://www.riaca.win.tue.nl/products/openmath/lib/1.3/api`, (2001-2004).

[13] *Document Object Model* , `http://www.w3.org/DOM/`

[14] Java Standard Edition, Sun Microsystems Inc, `http://java.sun.com`, (1995-2004)

[15] The Apache Software Foundation, `http://httpd.apache.org`, (1996-2004)

[16] Apache Jakarta Project, Apache Software Foundation, `http://jakarta.apache.org/tomcat`, (1999-2004)

[17] Apache Axis, Apache Web Services Project, `http://ws.apache.org/axis`, (1999-2004)

[18] Apache Ant, Apache Software Foundation,`http://ant.apache.org`, (2000-2004)

[19] JavaServer Pages Technology, Sun Microsystems, Inc. `http://java.sun.com/products/jsp`, (1999-2004)

[20] Maple, Maplesoft, a division of Waterloo Maple Inc. 2004, `http://www.maplesoft.com`, (2004)

[21] Mathematica, Wolfram Research, Inc., `http://www.wolfram.com`, (2004)

[22] Axiom, Software Foundation, Inc., `http://savannah.nongnu.org/projects/axiom`, (2000-2004)

[23] Derive, LTSN Maths, Stats & OR Network, `http://www.chartwellyorke.com/derive.html`, (2000-2004)

[24] MatLab, The MathWorks, Inc, `http://www.mathworks.com/products/matlab`, (1994-2004)

[25] The MBase Mathematical Knowledge Base. `http://www.mathweb.org/mbase/`