XML in Mathematical Web Services

Mike **Dewar**

Elena **Smirnova**

Stephen M. **Watt**

Abstract

We describe a framework for delivering mathematical web services, resulting from the MONET project [http://monet.nag.co.uk/cocoon/monet/index.html].

Mathematical problems often have a well-defined structure that can be described in detail at a fine-grained level. In the MONET architecture, mathematical services describe their capabilities using a Mathematical Services Description Language in addition to WSDL. When presented with a problem, a mathematical service broker selects an appropriate set of services and combines them to provide a solver.

We describe how two XML-based data formats, OpenMath and Content MathML, are used in a mathematical service toolkit based on the Maple computer algebra system. This service toolkit is based on a configuration engine that provides the appropriate conversions between the mathematical XML data formats, builds the necessary Maple program, and installs the necessary extensions to a generic Web services engine.

Table of Contents

1. Introduction

1.1. Motivation

Mathematics presents the perfect example of a richly structured, well defined problem domain. This contrasts with many problem areas where the exact meaning of a question is always subject to interpretation, or where the clear-cut questions are rather simple. In the domain of mathematical computation, problems in a rich context typically have a clear meaning and a well defined correct answer. The manner of arriving at the answer, however, is often difficult to determine: there may be several algorithms with different efficiency trade-offs, the algorithms may have their only implementations in different, incompatible software systems, or it may be necessary to combine algorithms that handle special cases for a particular problem. The objects involved in these computations may either be simple numerical quantities or be elements of elaborate mathematical models, with many semantic relationships and dependencies among the parts. From this point of view, mathematical computation provides a vantage point from which we can foresee problems that will later arise in other areas of semantic web services.

Mathematical web services are interesting from a purely practical perspective for several reasons. Foremost, mathematical algorithms often do not have many implementations. For example, there are optimization methods available only in particular numeric libraries; there are methods for decomposing multivariate systems of equations available only in particular computer algebra packages; there are methods of computing definite integrals that are available only in other, incompatible, computer algebra packages; there are asymptotically fast arithmetic functions only available in certain C libraries; there are computations in group theory and integer sequences that require access to special-purpose databases and use custom programs. No system implementor has the resources or the ability to integrate all these methods into a single package. From this point of view, mathematical web services can provide protocols allowing complex problems to be solved in a necessarily distributed and heterogeneous environment.

Mathematical web services also have a practical benefit from the economic perspective of a single user: with a vast array of mathematical software covering different, often overlapping areas, it is impractical to maintain local versions of all the packages a particular user might find useful. Although many of the packages are available without license fee, others have a substantial cost that cannot be justified for only occasional use for some unique functionality. Even maintaining an up-to-date collection of "free" packages involves substantial labour.

Finally, exploring mathematical web services helps us understand some of the complex issues that will arise in other areas: we have already observed that this well defined problem domain can serve as a bellwether for web services in general. They can also help us understand some of the emerging problem areas of grid computing, as they begin to require a broader range of mathematical techniques. Further into the future, we see that as mathematical software systems become more complex, with numerous independently written components, it is important to understand the potential for service discovery protocols and self-organizing behaviour even within a single system.

1.2. Objectives

From a high-level perspective, the overall objective of our work has been to understand the problems that arise in providing mathematical web services. This involves a number of related problems, including:

- How can mathematical services be provided most effectively in a platform-neutral manner, while preserving the semantics of the mathematical enquiries?
- How can mathematical services be adequately described, so that their functionality is precisely conveyed at a sufficient level of detail?
- What is required to compose services and to plan the solution to a complex problem?
- How can the solution of a complex problem be documented so that the result can be trusted, or that the responsibility for correctness be ascertained?

- How can this be achieved in a manner compatible with existing web technologies?
- How can this be achieved without requiring mathematical software specialists to also become specialists in web technologies?

Each of these problems is, in itself, broad in scope. In this paper, we concentrate on the role XML can play in each of these areas.

To make this investigation concrete, and as a practical objective of our work, we have had the additional goal of developing a prototype framework, populated with sample elements for each of its components. This has been undertaken as a collaboration within the MONET project, funded by the European Union.

1.3. The role of XML

This paper describes the role that XML has played in our framework prototype. We use XML to exchange mathematical data, to describe services and results, and to configure components. This involves a number of standard XML data formats and tools:

- *OpenMath*, a standard XML-based representation for semantic mathematical expressions, is used as the platform-neutral representation to exchange mathematical data in problem specifications and results. OpenMath provides an XML syntax for mathematical objects, as well as an XML format for content dictionaries describing the objects semantics.
- *The Web Ontology Language* (OWL) is used to build a universe of problem and solution method concepts.
- *XSL Transformation Language*(XSLT) is used for conversion between various data formats.
- *Content MathML*, another standard XML-based representation for mathematics, can be negotiated as a data transmission format between components. Additionally, it is used internally in the implementation of some components

In addition, we have defined a number of application-specific XML formats that play roles within the architecture:

- *Mathematical Services Description Language* (MSDL) is used by mathematical software components to publish detailed information about the services they provide. WSDL is also used to describe the format of the messages that they send and receive.
- *Mathematical Problem Description Language* (MPDL) is used to specify the mathematical problems to be solved
- *Mathematical Explanation Language* (MEL) is used to explain the process by which a solution has been obtained.
- *Mathematical Service Query Language*(MSQL) is used by clients to send problem instances with mathematical content to a broker.
- *Mathematical Services Configuration Language*(MSCL) is used to configure mathematical service engines. It provides the interface between the web-service and an application program, which need not be web-aware.
- *OpenMath to Maple Mappings*(OMM) allow modular definitions for translation between Maple and OpenMath.

The role of each of these is described in more detail throughoutthe paper.

1.4. Related work

We have previously reported on the MONET architecture in [**MKM04a**]and [**WSJ**]. Detailed descriptions of aspects of this work have been published separately. Work on the MONET ontologies has been reported in [**EW02**] and [**MKM04b**]. Service description and configuration has been described in [**IAMC04**]. Progress on mathematical service

discovery has been reported in [**AHM05**]. A related current project at the University of Manchester is "Instance Store" [**IS**]. A current project on mathematical service brokers at the University of Linz is [**MathBroker**]. In addition, experience in the MONET project has had a direct influence on two recent standards [**OWL**][**OM2**], to which members of the consortium have contributed. Earlier, related work influenced the form of [**MathML**].

1.5. This paper

- This paper examines the role of XML and related technologies in the provision of mathematical web services. We see the main contributions of this work as:
- a prototype architecture for mathematical web services, providing insight into all of our main questions;
- an implementation, testing these ideas, and providing useful demonstrator services;
- a demonstration that mathematical web services can effectively be provided and composed in a platform-neutral architecture using standard XML formats;
- a general, extensible strategy for conversion between OpenMath and application specific language, in this case Maple; and
- a mechanism to structure mathematical services so that application developers need not have any knowledge of web technologies.

The rest of the paper is organized as follows. Section 2 gives an overview of existing XML and Web Service technologies used as a base for the MONET project. Section 3 presents the overall architecture of the broker-based MONET framework for mathematical web services. Section 4 shows how XML is used in different parts of this architecture. Section 5 outlines some of the techniques for mathematical data translation between XML formats and their native representation in mathematical systems. Section 6 illustrates the ideas for the MONET architecture with an example of a symbolic mathematical web service. In Section 7, we analyse the global role of XML in the framework for mathematical web services. Section 8 summarizes three closely related XML-based projects, complementing the current project. We present our conclusions in Section 9.

2. Background technologies

At the starting point of our investigation, there were certain existing technologies that were related to our problem domain. *OpenMath*[**OM1**] and *MathML* [**MathML**] were two pre-existing XML formats for mathematical data exchange. *Owl*[**OWL**] was an emerging language for ontologies to be used in a web context.

It was clear that any experiments would require sample symbolic and numeric mathematical services. For these, *Maple* [**Maple**]was chosen as a functionally rich computer algebra system and the *NAG Libraries*[**NAG**] were chosen as an example of a complex suite of numerical software. Additionally, we used a number of standard tools for providing general web services. We describe each of these briefly.

2.1. MathML

MathML was developed under the aegis of the World Wide Web Consortium as a format to represent mathematics on the web and in other digital documents. It provides support both for the way mathematics should appear, through Presentation MathML, and for its semantics, through Content MathML. Presentation MathML provides a full range of elements to describe most mathematical notations. Because the range of mathematical semantics is much greater than the range of conventional notation, Content MathML was necessarily restricted in scope. Content MathML provides elements to convey the meaning of only elementary mathematical concepts. Content MathML can, however, reference externally defined concepts, notably including those defined by OpenMath.

2.2. OpenMath

OpenMath is a mathematical data exchange protocol devised to allow mathematical software packages to work together. Its origins date to the era when Maple and MatLab were first connected by an *ad hoc* protocol, and a more systematic solution was desired. The central concept in OpenMath is that of a *Content Dictionary*, or *CD*. Users, applications or groups can define CDs to specify the meaning of mathematical objects. For example one CD specifies the meaning of the trigonometric functions, while another specifies the meaning of statistical operations. Mathematical data is conveyed as expressions (e.g. in an XML syntax) with every symbol having reference to its defining CD. This inherently open-ended nature of OpenMath is at once its greatest benefit and its greatest weakness. Cooperation is required to agree on the set of CDs for any given discourse, and the set of CDs is ever-evolving.

2.3. OWL

The Web Ontology Language OWL [**OWL**] is a semantic markup language, endorsed by W3C, for publishing and sharing ontologies on the World Wide Web. In its reference implementation it uses the XML syntax of the Resource Description Framework (RDF), while its semantics draw heavily on previous work in Description Logics, in particular the design of the language DAML+OIL [**DAMLOIL**]. OWL models knowledge in terms of individuals, classes, and relationships, in a manner not dissimilar to that of an object-oriented programming language such as Java. Description Logics are decidable fragments of First Order Logic and, from a practical point of view, they have quite precisely-defined semantics which allow automatic reasoning to be performed over them.

2.4. Maple and NAG Libraries

Maple is a general purpose computer algebra system, developed originally at the University of Waterloo, and now developed by Maplesoft Inc. It has a comprehensive range of mathematical functionality, providing tools to manipulate equations and solve problems in most areas of applied mathematics. For example, Maple has facilities to provide symbolic solutions to integrals, differential equations, recurrences and a host of other areas. It is based on a kernel/library structure, where a small portion of the system is written in low-level C, and the majority of the mathematical functionality is provided by packages written in its own interpreted language.

The NAG Libraries are the most extensive collections of numerical algorithms commercially available in Fortran. They comprise nearly 1,500 routines covering a broad range of numerical and statistical areas. For example, they provide functions for optimization, solving ordinary and partial differential equations and linear algebra , as well as regression analysis, analysis of variance, time series and non-parametric statistics.

2.5. Tomcat, Axis, SOAP and JavaServer Pages

In this work we have used certain standard tools used in building web services. These include *Tomcat, Axis, SOAP* and *JavaServer Pages:*

Apache Tomcat [http://jakarta.apache.org/tomcat/] is the servlet container we used as a web application server to host MONET Mathematical Services. The services we have developed run under Tomcat, wrapped in the Axis servlet. Additionally, service web clients, implemented through Java Servlet Pages also managed by Tomcat.

Simple Object Access Protocol [http://www.w3.org/TR/soap/] (SOAP) is a light-weight XML-based protocol for de-centralized, distributed environments. In this project, SOAP is used to exchange messages between various parts of MONET architecture.

Apache Axis [http://ws.apache.org/axis/] is an implementation of the SOAP submission to W3C. In our project, it acts as a web servlet running within Tomcat and is used to wrap all of the mathematical services we have developed.

JavaServer Pages [http://java.sun.com/products/jsp/] (JSP) technology allows the creation of dynamic web content in a fast and convenient manner. We used it to design human client-end web interfaces to direct service calls.

3. Architectural overview

We now give a brief overview of the general MONET architecture for distributed mathematical web services. Aspects of the architecture are described in more detail in [**WSJ**] and [**MKM04a**].

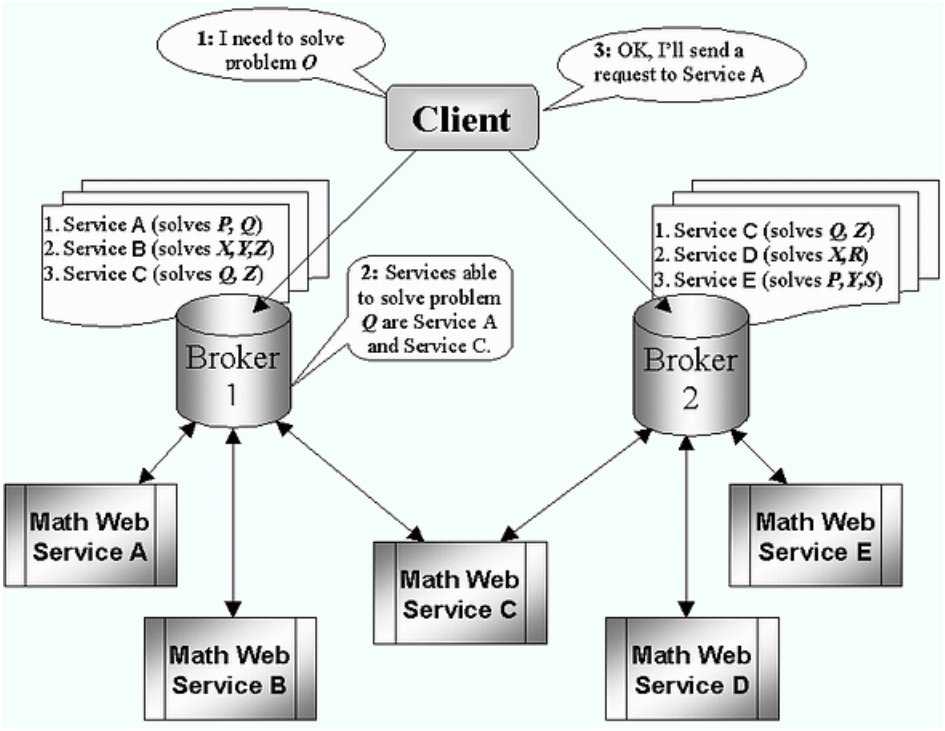
3.1. Components of the MONET framework

The MONET architecture comprises a set of communicating components in three different roles: *clients*, *services* and *brokers*.

Servers provide mathematical functionality, and expose these services by registering with brokers. A client sends a query to a broker, asking whether a web service is available to solve a specific type of mathematical problem, as shown in Figure 1.

In the basic case, the broker resolves the client's query by finding a suitable mathematical service and forwarding the client's request to it. The service then solves the mathematical problem. Finally, clients obtain the result of the computation and, optionally, a meaningful explanation of how it was arrived at.

In a more complex case, a broker could break down a complex service request into steps and use a sophisticated planning process to compose multiple services. However, from both the client's and the service's perspective, the nature of the interaction remains the same.



Clients send requests with problem descriptions to brokers that, in turn, try to find an appropriate mathematical service to solve the problems.

Figure 1. General scheme of MONET architecture

3.2. Mathematical Broker

The task of the broker is to match the characteristics of a given mathematical problem to the best Service available, or if no single service can solve the problem by itself, to divide the problem into sub-problems that can be solved by the services available.

It is assumed that the broker is a sophisticated search engine and planner for complex mathematical problems. In general, matching requests for arbitrary mathematical problem is a hard to implement. However, the MONET strategy for service matching and discovery seems to be capable of providing a solution for a wide range of problems that appear in practice, as has been shown over the course of the MONET project trials. The matchmaking technique has been discussed in a number of papers, including [**MKM04b**]and, more recently, [**AHM05**].

3.3. Mathematical Services

In the MONET architecture mathematical web services are provided as black boxes without revealing their implementations. The manner in which the mathematical service is provided is opaque to the client: it may be provided by a stand-alone C program, by a call to a computer algebra system, or even by a human being! However, from the point of view of external clients and internal server tools, all of the services within the framework must have consistent interfaces.

This approach ensures flexibility and extensibility of each service individually and the whole framework in general. It supports changing the implementation of a service, for example switching between the underlying mathematical systems, and allows for the easy creation of new services.

We have designed an automated environment to run mathematical web services powered by the Maple and Axiom computer algebra systems. This environment provides tools for automatic service creation, installation and invocation. As will be shown in the next section, each mathematical service is created from a single XML file, provided by a service author. The rest of the process of service deployment and operation is supported by the mathematical web server environment. Among its tasks is provision of the connection with the mathematical backengine, including creating Maple or Axiom programs on the fly, based on the information in the service configuration file. Exactly the same approach could be used with any other computational software package.

4. Use of XML

This section describes how several XML formats are used in different parts of the MONET framework. There are several communication streams between the client, broker and server components. Certain of these communications use what have already become standard XML formats, such as OpenMath, MathML and OWL. Others have required the definition of new XML-based formats:

The query sent from client to Broker, includes

a. general problem description, in Mathematical Problem Description Language,

b. the expressions giving the actual mathematical problem content, in OpenMath.

A SOAP message sent from Broker (or Client) to a Math Web Service encapsulates

a. mathematical problem expressions, in OpenMath,

b. optional meta-information, such as the type of result required, explanation verbosity level, etc.

The response from the Service back to Broker or Client includes

a. the result of the calculation or proof, in a format such as OpenMath or MathML,

b. an explanation of the result, in Mathematical Explanation Language.

4.1. Ontology creation front end

4.1.1. OWL and Mathematical Services Description Language (MSDL)

The Web Ontology Language (OWL) [**OWL**]allows us to define URIs to refer to particular objects and concepts, and to represent different kinds of relationships between them. In the MONET framework OWL is used to represent many different kinds of concept including mathematical problems, hardware and software platforms, algorithms and academic papers. This is then used in the service matching process, to allow queries such as

- find a service which computes definite integrals over a finite range
- find a service which uses the algorithm described in the paper by Bronstein in the proceedings of ISSAC 1991
- find a service solving differential equations using the NAG Library
- find a sparse solver running on a machine with at least 64 processors.

There are three aspects to this process. The first is that we must build a set of ontologies, which is rich enough to express all the concepts that we are interested in, and identify a mechanism for reasoning over it. The second is that we must describe all the available services using these ontologies. The third is that we must have a mechanism for describing the user's query.

In principle we can express all of these things using one of the OWL syntaxes, and use one of the available OWL editing environments (such as Protege [**Protege**]) to author them. In practice this is not very attractive: it requires people deploying services or creating queries to understand the subtleties of OWL and use very verbose syntax. Instead we created a simple XML application, the *Mathematical Service Description Language* (MSDL), which allows somebody using the MONET framework to make statements about the services they are deploying or the problem, which they want to solve. Documents written in MSDL can be validated against the MSDL schema and then translated to OWL using appropriate XSL stylesheets. Full details of this process can be found in [**MKM04b**]

Mathematical Problem Description Language (MPDL)

4.1.2. Mathematical Problem Description Language (MPDL)

The Mathematical Problem Description Language (MPDL) is a subset of MSDL which allows a user to describe a problem in terms of its inputs and outputs, pre-conditions (i.e. mathematical relationships between the input objects), and post-conditions (relationships between the input and output objects). There is an obvious similarity to the way that the behaviour of components in process execution languages are described, but in this case our principle aim is to describe the mathematical nature of a problem in an unambiguous way. The descriptions are grouped into a library, each member of which has a unique URI. The OWL ontology allows us to refer to these problems, and describe their relationship with other concepts. For example, given a problem *P*, a software system *S*, and an algorithm *A*, we can express the fact that *S implements A which solves problems of type P*.

For example the problem of finding the minimum value of an expression subject to simple bounds on its parameters where both the expression and its derivatives are present might be expressed as follows.

- **Inputs**
  1. $F(v_{1,..,}v_n)$: $R^n \rightarrow R$
  2. $A \subseteq R^n$
- **Outputs**

1. $\{D_i(v_{1,..,}v_n)\colon R^n \rightarrow i = 1..n\}$
2. $x \in A$
3. $f \in R$

- **Pre-conditions**

1. $D_i = \partial F/\partial v_{i,}\ i = 1..n$

- **Post-conditions**

1. $F(x) = f$
2. $\neg\exists\ y\ \forall\ A\colon F(y) < f$

One important use of such a description is to provide names for the various objects that form parts of the problem (in this case *F, A, D_i, x* and *f*). They can then be used in formulating a problem (i.e. one can say *F = x*sin(x)* etc.). They are also useful in describing how to take the objects provided by the user, and construct an XML message of the format required by a web service (as described in its WSDL file).

4.1.3. Mathematical Service Query Language (MSQL)

The Mathematical Service Query Language defines the format of queries that clients submit to MONET brokers. Each MSQL message consists of mathematical and logistical parts.

The mathematical part includes a description of the mathematical problem, expressed in MPDL, and the content of the mathematical problem given in OpenMath. In the example of Section 4.1.2, *F* is a placeholder for a function in a problem description, while *x**sin(*x*) could be the mathematical content corresponding to *F.*

The logistical part may contain optional information about the client, constraints on the expected results or constrains on the set of acceptable mathematical services. For example, the client may define mathematical software of the service to be *Maple 10* or specify the type of hardware for the mathematical server.

4.1.4. Mathematical Explanation Language (MEL)

In scientific computing, and indeed in many other areas of information technology, two important issues are *reproducibility* and *provenance*. Reproducibility boils down to the idea that if you give somebody your data and they perform the same experiment as you, they should get the same results. Provenance is concerned with keeping a record of where a set of data came from originally and what has been done to it since it was first collected. In a framework such as MONET's, a user would like to know what services have been used to solve his or her problem, what data they received and, perhaps, how they processed that data internally.

4.2. Mathematical markup languages

The MONET architecture makes use of two standard XML-based mathematical markup languages to convey mathematical expressions, OpenMath and MathML. We have already described these briefly, and now describe more precisely how they are used within the architecture.

4.2.1. OpenMath

The OpenMath format is used to represent *all* mathematical components of messages controlling the MONET architecture. This includes the mathematical expressions within the Mathematical Problem Description Language, and any

mathematical expressions that might appear in a Mathematical Service Description. It is also used to represent mathematical expressions in client queries and broker planning.

Additionally, unless otherwise specified, OpenMath is used to communicate the mathematical content of the actual problem instance from the client, and the mathematical content of the result back from the server.

4.2.2. MathML

MathML is a second, optional format that clients and servers may use to communicate the mathematical content of problem instances and results. Only Content MathML may be used for the mathematical content of a problem instance, but any combination of Content or Presentation MathML may be returned from a service, if required.

MathML is also useful for viewing mathematical content in web browsers. We have built a set of XSLT stylesheets to convert OpenMath to Content MathML, and Content MathML to Presentation MathML [**MathML02**] which may be used for this purpose.

4.3. Mathematical service configuration

It is not surprising that the services are described using XML-formats. One well-known XML-based language for presenting interfaces of general web services is WSDL **[WSDL]**. For mathematical web services, we have found it necessary to enrich the service description capabilities by introducing a Mathematical Services Description Language (MSDL), described above, and a Mathematical Service Configuration Language (MSCL), described here.

4.3.1. The purpose of configuration files

A service configuration file provides the connection between a service's interface and its actual implementation. This file is the single component needed in the framework to represent a service. It includes information about the service MSDL, service interface to the mathematical engine and an invocation of the service implementation..

This implies that to create a new mathematical service one need only write its configuration file. More precisely, one need only write a skeleton of a configuration file, which will be completed automatically during service installation. All that is required from the author of a service is to provide the actual service implementation (a program in the language of the underlying mathematical solver), along with an entry point to this implementation (an interface function call). The language of these configuration files is MSCL.

4.3.2. The structure of the configuration file

In general, service configuration file may contain descriptions for more than one service. This makes sense when several services share parts of their implementations.

It is also sometimes convenient to specify alternative service implementations for different mathematical software packages. For example, for indefinite integration one could use Maple, Axiom or Derive. In this case, a configuration file would contain multiple entries for the service implementation, each of which is marked as being for the corresponding mathematical solver.

The implementation part of a mathematical service configuration may be straightforward or very elaborate. The implementation may be as simple as a single call to a built-in function of the software package. More often, providing the service requires a certain amount of custom code to be written. This can either be included in the implementation part, or saved in a library that is referenced from the implementation part. If it is included, then the implementation part can be quite lengthy. If it is referenced (loaded) by the implementation part, then the implementation will be very short.

The markup below presents the general structure for a mathematical service configuration file.

```
<mathServer>
   <msdl>
      <service name="service_A">
           {MSDL service A}
      </service>
      ...
      <service name="service_Z">
           {MSDL service Z}
      </service>
   </msdl>
   <services>
      <service name="service_A" call="interface_call_for_sevice_A"/>
      ...
      <service name="service_Z" call="interface_call_for_service_Z"/>
   </services>
   <implementation language = "math_solver_1">
      {implementation of services interface calls in math solver 1}
   </implementation>
   ...
   <implementation language = "math_solver_N">
      {implementation of services interface calls in math solver N}
   </implementation>
</mathSever>
```

4.3.3. The benefits of using configuration files

The strategy of representing mathematical services by their configuration files significantly eases service creation and maintenance.

To provide a new service, authors need only know their preferred mathematical software package and do not need to be web service experts. In particular, they do not need to know Java or WSDL. Once described in the configuration file, a service is deployed using tools from our Mathematical Web Service Environment. This configuration file is portable, and may be used to install the service on any server running the Mathematical Web Services Environment. Most of the service properties and functionality can be modified without service re-installation: it is only necessary to update the configuration file.

We note explicitly that a service provider can offer the same service based on a variety of mathematical software packages.

5. Translation to and from implementation data format

While the mathematical content of problems is usually communicated in OpenMath or Content MathML within the MONET architecture, clients and servers do not usually use this format internally, and it is necessary for them to convert data to and from these formats. We illustrate how this may be done by detailing the conversion between OpenMath and Maple expressions.

Two main issues arise in these conversions. The first is that in OpenMath each symbol has a unique meaning, specified by an explicit content dictionary, while in Maple symbols can be overloaded and ambiguous. The second issue is that OpenMath is inherently extensible, so any general approach to translation must be able to handle translation with respect to problem-specific content dictionaries.

Note that the approach described in this section is based on the pre-existing XML infrastructure in Maple. If a mathematical package does not have adequate XML support, then either it must be added or the translations must be done by an external program.

Conversion between Maple and Content MathML could be done *ab initio* in a similar way. (Note that the Content MathML may contain references to externally defined quantities, possibly specified by OpenMath content dictionaries.) In practice, however, we have found it easier, when going from Content MathML to Maple, to convert Content MathML to OpenMath and use the package described here. The other direction is straightforward, as Maple natively supports an export to Content MathML.

5.1. Naive approaches

The first approach we considered for translation between Maple and OpenMath is to use XSLT. The conversion from OpenMath to Maple would use one stylesheet to produce Maple source code. The conversion from Maple to OpenMath would use a two-step process, first using Malpe's export to Content MathML, and then using an XSLT stylesheet to convert from Content MathML to OpenMath.

This approach has two problems:

1. The main problem is that OpenMath and Maple are both extensible. Any service using any new OpenMath CD, or new Maple functionality, would have to produce use modified versions of these XSLT stylesheets for translation. There could be no one fixed program that handled the translation.

2. The second problem is more subtle. With two stylesheets handling the two conversion directions, it is difficult to maintain consistency in the bidirectional correspondence between Maple and OpenMath expressions.

The second approach that we considered was to use another technology, Java, to do the conversion. This was particularly tempting since a partial OpenMath/Maple translation was already available. This approach, however, had the same problem as the XSLT solution: working in any new area would require the extension of the translations. This would require the mathematical service programmer to understand how to extend the Java components of the architecture, and also distribute knowledge of Maple throughout the architecture's components. Both of these consequences were clearly undesirable.

5.2. Extensible table-based transformations

The approach we eventually settled on was to perform a direct translation between Maple and OpenMath expressions. The conversion is carried out by a Maple program, driven by a collection of bi-directional mapping files. New OpenMath Content Dictionaries can be handled by adding new mapping files.

5.2.1. Approach overview

It is often the case that OpenMath data has patterns that correspond directly to Maple data structures, as illustrated in Table 1, “Similarity between OpenMath and Maple patterns”. This fact has motivated us to organize a converter as an engine driven by a set of mapping files describing the correspondence between these patterns. The scope of mathematical content that the converter can handle is determined by the set of mapping files it has loaded. This allows us to configure the converter vocabulary, which is necessary to support OpenMath's extensible vocabulary.

OpenMath	Maple
<OMOBJ> <OMA> <OMS cd="transc1" name="sin"/> <OMV name="x"/> </OMA> </OMOBJ>	sin(x)

Table 1. Similarity between OpenMath and Maple patterns

5.2.2. Mapping file design

We wish to support an organization with one mapping file per OpenMath Content Dictionary, and to use it to drive both directions of the conversion. An entry in a mapping file contains a fragment of OpenMath markup and the corresponding form of a Maple command. Each entry in a mapping file has to take into account type, position and number of arguments, as shown in the listing.

The choice of pattern language was selected to support rules for most of the common cases, rather than for every possible correspondence. It was simplest to have an easy pattern language, and to handle the unusual cases with Maple helper functions. Many of Maple expressions in the form `head(arg_1,arg_2,...)`can be transformed directly into OpenMath in the form `OMS(arg_1, arg_2 ...)` and vice versa. For those cases conversion rules are straightforward, and can be defined by mapping templates similar to those shown in Example 1, “Sample entry in the mapping file”.

Example 1. Sample entry in the mapping file

```
<map>
 <OpenMath>
    <OMA>
       <OMS cd="transc1" name="sin"/>
       <code>
          e -> [MobToOpenMathConversion:-clean_uneval_then_do(op(1,e))];
       </code>
    </OMA>
 </OpenMath>

 <Maple>
    <head arg_num="1">sin</head>
    <Maple_args>
       <arg/>
    </Maple_args>
    <OM_args>
       <arg pos="1"/>
    </OM_args>
 </Maple>

</map>
```

The conversion of expressions is performed from top-down. When converting from Maple to OpenMath, special care must be taken in evaluation to preserve the original form of the expression. For example, if we try to convert the Maple expression `sin(0)^2+cos(0)^2 = 1` to OpenMath, instead of the expected result we would get an OpenMath

encoding of the expression `"1=1"`. To avoid this situation, a special function must be called to delay evaluation in arguments of `<OMS>`, as shown in the above example.

5.2.3. Challenges and solutions

Not everything can be handled by the simple form of rule we have seen. Differences in the design of the OpenMath and Maple languages, the forms of certain expressions, and the dynamic typing in Maple all introduce problems with this approach.

First, not all OpenMath to Maple mappings can follow the simple pattern of the previous example. This is simply because of some significant difference in the mathematical expression layout, beyond the pattern description capacity of the mapping file. This occurs, for example, with partial differentiation: the operation in Maple takes a list of differentiation variables, e.g. `diff(f(x,y,z),x,z)`, while the corresponding OpenMath expression uses indexes to achieve the same purpose, e.g. `partialdiff(list(1,3), f(x,y,z))`.

Second, there are some OpenMath symbols that do not have any equivalent in Maple syntax. For this case, we throw an error, reporting that these OpenMath symbols are not supported. We should not encounter this in practice, because if Maple cannot represent the symbol, then it cannot perform the service, and this should have been apparent to the broker.

Our mapping file design provides a solution to this difficulty, by allowing snippets of translation code to be included in the mapping file when necessary. This is done by replacing the `<args>` element with a `<code>` element containing a Maple translation procedure. Example 2, “Mapping template for a special case.” shows how to translate the `log` function. A call to the Maple function `translate` will recursively continue transforming subexpressions, using the same set of mapping files, now referred to by the `translationTable` parameter.

Example 2. Mapping template for a special case.

```
<map>
    <OpenMath>
      <OMA>
        <OMS cd="transc1" name="log"/>
        <arg pos="1"/>
        <arg pos="2"/>
      </OMA>
    </OpenMath>
    <Maple>
      <head>log</head>
      <code>
        proc(expr)
          log[op(translate(op(1,expr),translationTable))]
          (op(translate(op(2,expr),translationTable))):
        end proc:
      </code>
    </Maple>
 </map>
```

The dynamic typing of the Maple language introduces another problem: the interpretation of mathematical expressions is open to ambiguity. In general, the meaning of an expression cannot be determined by the name of the applied function form alone. For example, Maple uses the same function call `int` for both definite and indefinite integration. In contrast, OpenMath has two distinct symbols for these operations: `int` and `defint`.

Some ambiguities can be resolved by inspecting the number and types of arguments. For ambiguities that cannot be resolved in this manner, we employ one of two techniques:

1. The first technique is to resolve the ambiguity with a Maple disambiguation procedure, supplied as a parameter when the converter is invoked. An example strategy for the disambiguation procedure is to choose the first possible OpenMath correspondence found. We also can to assign priorities to mapping entries through a `prior` attribute of the `<map>` element. We can then follow the conversion pattern given by the `<map>` entry with the highest priority.

2. The second technique is to generate a pseudo-OpenMath representation that can be dealt with later. In some cases a structure of a pseudo-OpenMath expression can be refined based on context information of neighbouring expressions.

In the worst case, an error is thrown notifying the invoker that the ambiguity cannot be resolved.

5.2.4. Current status

Presently, mapping files for forty-six Content Dictionaries are implemented. Of the twenty-eight Content Dictionaries in the MathML-compatibility CD group, all but three are implemented. The rest of the Content Dictionaries are experimental, i.e. their contents have not yet been fixed by the OpenMath society.

6. A detailed example

We give an example of a MONET mathematical web service for symbolic order differentiation. This problem is a generalization of ordinary differentiation, computing derivatives of symbolically defined order. For example the service can compute an expression for the *n*-th derivative of tan(*x*), where *n* is a symbol standing for an un-specified positive integer. This, and a number of other demonstration mathematical web services are available at the ORCCA MONET website [http://www.orcca.on.ca/MONET].

6.1. Problem description

When introducing a new mathematical service, we must ensure that the service will be recognized by brokers. This requires specifying a set of problems that this service is able to handle. The formal description of the problem of symbolic order differentiation according to the MONET Mathematical Problem Description Language (Section 4.1.2) can be given as follows:

Problem

```
symbolic_order_differentiation
```

Header

Taxonomy: http://gams.nist.gov [http://gams.nist.gov/]

Code: GamsO

Body

- **Input**

  $f \in$ *algebraic expression*

  $n \in$ *algebraic expression*

- **Output**

  derivative $\in$ *algebraic expression*

- **Post-Condition**

  $\forall x. x \in \mathrm{R} \Rightarrow f(x) - F_n(x) = c_0\, c_1 x c_{n-1} x^{n-1}$

  where

  c_0, c_1, c_{n-1} are constants,

  $F_1(x) = \int (derivative)\, dx$,

  $F_k(x) = (F_{k-1})\, dx$, for k=2..*n*

For brevity we do not include here the explicit MPDL markup for the above problem description, but rather will assume that this markup exists and stored in at URI `http://aURI/symbolic_order_differentiation`.

6.2. Creating the service: configuration file

6.2.1. MSDL part

```
<msdl>
 <service name="SymbolicOrderDiffService">
 <classification>
    <taxonomy taxonomy="http://gams.nist.gov" code="Gams0"/>
    <problem href=" http://aURI/symbolic_order_differentiation "/>
    <format>http://www.openmath.org</format>
    <directive-type href="http://monet.nag.co.uk/owl#evaluate"/>
 </classification>
 <implementation>
    <software href="http://monet.nag.co.uk/owl#Maple9"/>
    <hardware href="http://monet.nag.co.uk/owl#PentiumSystem"/>
 </implementation>
 <service-interface-description href = ""/>
 <service-binding>
  <map operation ="op" action="action"
       problem-reference="http://aURI/symbolic_order_differentiation"/>
  <message-construction
         io-ref ="http://aURI/symbolic_order_differentiation#f"
         message-name ='%mn1' message-part='%[1]'/>
  <message-construction
         io-ref ="http://aURI/symbolic_order_differentiation#n"
         message-name ='%mn2' message-part='%[2]'/>
  <message-construction
         io-ref="http://aURI/symbolic_order_differentiation#derivative"
         message-name ='%mn3' message-part='%return'/>
 </service-binding>
 <service-metadata/>
 <broker-interface>
    <service-URI/>
    <broker-interface-description/>
 </broker-interface>
 </service>
</msdl>
```

The MSDL creation assistant forms the *MSDL skeleton* part of a configuration file, where the author of the service had to specify

- the problem classification with a reference to a problem description;
- general information on service implementation, such as hardware and software;
- some details on the number and type of arguments to the service, including an important part about the binding between service arguments and input parameters, specified in the problem description (note symbols prefaced by '#' in the above listing).

The rest of the data, for example the actual service URI or location of its WDSL, cannot be determined until the service is installed and deployed on the server. In the listing above, all entries of the MSDL skeleton that should be dynamically defined are prefaced with '%'. Their actual values will be assigned automatically by the service installation manager. The completed version of MSDL descriptor will then be used by the same service installation manager to register the new service with math brokers.

6.2.2. Service interface to the math engine and implementation

These two parts are to be completed by the author of the server manually. For someone who is familiar with Maple, or any other mathematical software package usable by the Mathematical Web Server, these steps are the least difficult part of service creation.

In the case of simple services, such as ordinary differentiation or integration, authors can simply delegate service interface calls to standard Maple procedures. For more sophisticated services, authors may provide their own code or give a reference to an external library or package. In both cases, the second and third part of the configuration file will look similar to the one for this example.

Service interface to mathematical software is given by the following entry to the configuration file:

```
<services>
<service name="SymbolicOrderDiffService"
            call="SymbolicOrder:-Differentiate"/>
</services>
```

The actual service implementation with Maple appears under the element `implementation` specifying "maple" as the value of the `language` attribute. When this service was originally defined, symbolic order differentiation was not provided by Maple. Therefore the implementation part contained the definition of a supporting Maple package:

```
<implementation language = "maple">
# package SymbolicNthDerivative
# author Dr.Robert Corless
# date Winter 2004

SymbolicOrder := module()option package;
   export Differentiate, Integrate;
   Differentiate := proc( f::algebraic, x::name, n::algebraic )
        local x,y,z;
        # ... body of function omitted ...
        return derivative;
    end proc;
end module;
</implementation>
```

6.3. Calling the service

When a client requires a mathematical service to compute a symbolic order derivative, it first submits a MSQL query (see section 4.1.3) to a mathematical broker. The following query tells the broker to find a service that can compute d^p/dx^p(x*sin(x)).

```
<query id="id1" xmlns="http://monet.nag.co.uk/monet/ns">
    <classification>
        <taxonomy code="GamsO" taxonomy="http://gams.nist.gov"/>
        <semantics name="OpenMath" href="http://www.openmath.org"/>
        <directive-type>
            http://monet.nag.co.uk/owl#evaluate
        </directive-type>
    </classification>
    <monet:problem xmlns:monet="http://monet.nag.co.uk/monet/ns"
              name="http://aURI/symbolic_order_differentiation">
        <monet:header/>
        <monet:body>
            <monet:input name="f">
                <monet:signature>
                  <!-- ... OpenMath encoding of x*sin(x)... -->
                  <OMOBJ xmlns = 'http://www.openmath.org/OpenMath'>
                     <OMA>
                        <OMS cd = 'arith1' name = 'times'/>
                        <OMA>
                           <OMS cd = 'transc1' name = 'sin'/>
                           <OMV name = 'x'/>
                        </OMA>
                        <OMV name = 'x'/>
                     </OMA>
                  </OMOBJ>
              </monet:signature>
            </monet:input>
            <monet:input name="n">
                <monet:signature>
                    <OMOBJ>
                        <OMV name="p"/>
                    </OMOBJ>
                </monet:signature>
            </monet:input>
            <monet:output name="derivative"/>
        </monet:body>
    </monet:problem>
    <constraint id="id2" encoding="http://anexternalURI.com/encoding#A1">
       <!-- constraints on the solution or the service to be called -->
    </constraint>
 </query>
```

This query contains a problem description that a broker will match with the problem reference in the MSDL of the symbolic order differentiation service. Then, if there are no constraints in a client query against this, the broker will send an invocation call to our service. A special component of the broker, called the *Execution Manager* maps the mathematical data from the query to the service interface. Mapping rules are provided by the *service-binding* part of the service's MSDL description.

The OpenMath input of the problem will be serialized and wrapped in a SOAP message according to the service WSDL descriptor, and the SOAP message received by the service will cause its invocation. The Mathematical Web Services Environment will retrieve from the service configuration file the implementation part, and generate the Maple program:

```
restart;
 #!! Copied from Implementation part of configuration file

   # package SymbolicOrder
   # author Dr.Robert Corless
   # date Winter 2004
   SymbolicOrder := module()option package;
   export Differentiate, Integrate;
   Differentiate := proc( f::algebraic, x::name, n::algebraic )
        # ... body of function omitted ...
        return derivative;
    end;
  end; #module

  # .. details omitted ..
  omArgs:=[<OpenMath encoding for 'x*sin(x)'>,
           <OpenMath Encoding for 'p'>];

  # Convert OpenMath expressions Maple expressions
  mapleArgs:=seq(OpenMathToMobConversion:-omtomaple(omArgs[i]),
              i=1..nops(omArgs));
  #!! Using functional call from Interface part of configuration file
  output := SymbolicOrder:-Differentiate(mapleArgs):

  #.. details omitted ..
  ans:= MobToOpenMathConversion:-mapletoom(output);
  ans:= XMLTools:-PrintToString(ans);
  #.. details omitted ..
quit;
```

Note that the generated program contains the implementation provided by the mathematical service configuration file. It also contains statements to convert the arguments from OpenMath to Maple objects, and to convert the result back to OpenMath.

The service will encode the response from Maple along with some additional information in MEL format:

```
<query-response xmlns="http://monet.nag.co.uk/monet/ns">
  <executionResponse xmlns=http://monet.nag.co.uk/monet/ns
           href="http://orcca.on.ca/services/SymbolicOrderDiffService">
    <explanation>
       <errorcode>-1</errorcode>
       <explanationFormat isProvided="true"
                 href="http://monet.nag.co.uk/monet/explanation#expl1">
          <serviceExplanation xmlns="http://aURI/explain">
            <Algorithmn>
                SymbolicOrder:-Differentiate
            </Algorithmn>
            <ExecutionTime>
               7ms
```

```
            </ExecutionTime>
          </serviceExplanation >
        </explanationFormat>
      </explanation>
      <result>
        <resultFormat
                 href="http://monet.nag.co.uk/monet/result#openmath">
              <!-- OpenMath encoding of
                   x*sin(x1/2*n*Pi) - n*cos(x1/2*n*Pi)   -->
        </resultFormat>
      </result>
    </explanation>
  </executionResponse>
</query-response>
```

Axis will wrap the response from the symbolic order differentiation service in a SOAP packet, and send it back to the broker, and then on to the client.

7. Analysis of the use of XML formats

7.1. XML as a universal syntax

There is nothing particularly radical about the XML data model - it is virtually identical to that of Lisp which has been around since 1960 - however what makes it so useful in the context of a framework such as MONET is both its widespread acceptance, and the rich suite of tools and technologies which surround it. These have enabled us rapidly to combine pre-existing technologies with our own mechanisms resulting in a consistent, unified whole. In addition, the fact that XML is becoming the accepted *lingua franca* for importing and exporting data to and from applications, meant that software packages such as Maple could easily be adapted to work inside the MONET framework. We have made extensive use of XSLT [**XSLT**]to translate between the different formats used (for example to transform a service description expressed in MSDL into one expressed in OWL), and of the XML Schema Language [**XSD**]to ensure that at each stage of the process all our documents are syntactically correct.

The MONET Consortium was made up of seven organizations with developers working at eight different locations in three different time zones. Creating a coherent body of software with such a large, distributed team is obviously made easier if each team has its own, well-defined, piece of software to work on. The flexibility of XML and the ability easily to transform a document conforming to one schema to a document conforming to another made the process of integrating the different components much easier. It meant for example that the designers of the Mathematical Service Description Language could focus on making it easy to use by a service deployer, without worrying about how it would be transformed into an OWL document suitable for processing by a reasoner.

Of course, while syntax is important in a practical sense, semantics are crucial. The existence of OWL provided us with a natural, XML-based, language in which to express the precise semantics of the various components in the MONET framework.

7.2. XML as an abstract heap data model

One model for implementing web services is as *factories*, where for every application a private instance of the web service is instantiated. This is very useful because it allows each instance of the service to maintain its own state based on the history of transactions between it and its clients (a human user and/or other web services). The downside of this model is that it is very inflexible: in particular the service instance has to run continuously until its client decides that it is no longer needed. At the opposite extreme, a web service can be seen as stateless, with all the necessary context for solving a problem being encoded in the messages it receives from its clients. This of course increases the size of

the messages passed from client to service and as a result, in an application area where objects can routinely by several gigabytes in size, substantially degrades overall performance.

In MONET we decided to adopt a compromise approach to this issue. We like the flexibility that stateless services offer: for example the fact that we can switch the server executing a service mid-way through a sequence of transactions to optimize the use of hardware resources is very attractive. However we wanted to be able to store objects somehow so that we could refer to them via a URI rather than encode the whole object inside each XML message. The resulting software is the *Mathematical Object Manager (MOM)*.

The MOM is a distributed, hierarchical system. A service deposits a copy of an object in its local MOM and receives its URI in return. This local MOM propagates the fact that it has an object matching this URI to its parent which, in its turn, propagates this information on up the tree. Whether the object itself is also propagated is a configuration issue. When a service needs a copy of the object it requests it from its local MOM. If it does not have a local copy it propagates the request up the tree until it reaches a node which recognizes the URI. It may have a local copy of the object but, if it doesn't, it knows where the object can be found.

A nice consequence of this is that it offers an alternative model for asynchronous communication between services. A URI for an object can be issued before it has been computed and lodged in the MOM. This URI can be passed between services which will only block when they actually need to access the object it represents.

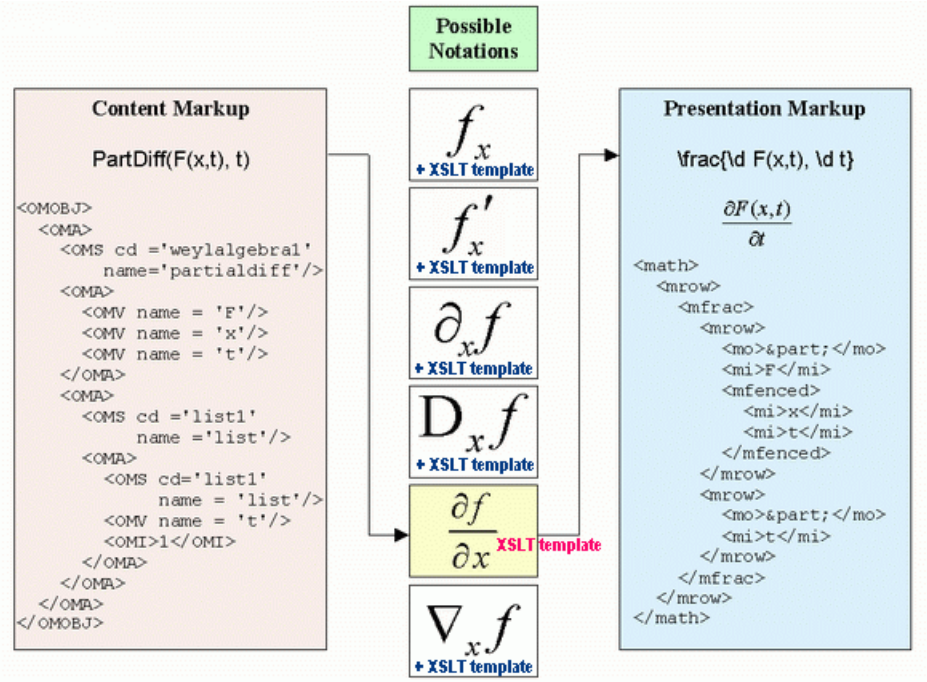
The objects in the MOM can be generated by any and every MONET application, each of which have their own internal format. One or more common formats are therefore needed for the objects and, not surprisingly, we chose formats based on XML, namely OpenMath and Content MathML. Should an application which supports OpenMath receive an object in Content MathML from the MOM it can always invoke a translation service to transform it into a format it is capable of handling.

8. Related uses of XML in mathematical web services

When using the computational facilities of mathematical web services, we often need to present mathematical expressions in a human-readable or browser friendly format. We have developed at ORCCA several additional XML-based tools that can be used for mathematical data transformations.

8.1. Notation selection tool

The Notation Selection Tool [http://www.orcca.on.ca/MathML/NotationSelectionTool] is designed to perform conversion of mathematical expressions in XML format: it allows the user to specify which of several mathematical notations should be used in mathematical applications, including mathematical web services, as illustrated in **Figure 2.** The purpose of the tool is twofold. First, it allows the user to set a preferred notation for rendering specific mathematical concepts, for example a user can select open intervals to be denoted (a,b) or $]a,b[$. Second, it helps to distinguish different mathematical concepts that may use the same notation, such as $\sin^{-1}(x)$.



XSLT templates assigned to different notations carry out the translation from content to syntax encoding.

Figure 2. Selecting mathematical notations

An XML-format initialization file is used to configure the Notation Selection Tool. This file contains a database of mathematical concepts, alternative notations, and XSLT templates to produce Presentation MathML for the selected notations.

Using an initialization file for the Notation Selection Tool provides it with a high degree of flexibility and extensibility. On one hand, it allows the user to introduce new notations for existing math concepts simply by editing the initialization file. On the other hand, new mathematical concepts can easily be introduced in existing settings. To illustrate the second case one can imagine setting notations for binomial, which does not appear in standard MathML (neither Content nor Presentation). However it can easily be introduced via an additional stylesheet template.

The same approach allows one to set preferred rendering for OpenMath CDs. More detailed information on the Notation Selection Tool organization and implementation can be found in [**NAMKM04**].

8.2. Content-faithful transformations

A standard strategy for displaying Content MathML or OpenMath in a browser is to do so by converting it to Presentation MathML. This is sufficient for simple display, but too much information is lost to support applications that involve selection, editing or manipulation.

To preserve the original meaning in the transformed expression, we have introduced the notion of "content-faithful" transformations. The basic idea of content faithful transformations is to provide sufficient information to be able to find the content associated with the output of a transformation. There are several ways to do this, including generating high-level parallel markup, fine-grained parallel markup, or parallel markup with fine-grained cross-linked subtrees. This is described in detail in the papers [**ORCCA00**][**MathML02**].

Content faithful transformations also allow the preservation of useful "Extended MathML" markup. By this we mean XML markup containing both MathML and non-MathML elements, where there is a stylesheet to convert the non-

MathML elements into some valid MathML, perhaps with annotations or external references. This is often convenient when working with concepts that are beyond the scope of Content MathML.

8.3. Semantics conserving TeX translation

A similar approach to conserving semantics is applicable to translation between XML-based and non-XML based markups. In the context of mathematical web services, one of the potential sources and ultimate destinations of mathematical expressions is in TeX documents. To support this, we have developed tools for mapping TeX to MathML and MathML to TeX. Our approach, described in [**TUG02**] and in more detail in [**IAMC02**], allows mappings to specify a correspondence between high-level elements in TeX and MathML markup. This way it is not necessary to expand macros fully, and it is possible to conserve the implicit semantics the macro markup conveys. Bidirectional mapping files (in an XML syntax) specify the correspondence between the extended TeX and the extended MathML markup. For example it is possible to define an XSLT stylesheet corresponding to a LaTeX class file, and to define a mapping file between the LaTeX macros and XSLT templates.

9. Conclusions

We have studied the problems arising in providing mathematical web services and have devised a prototype framework with a number of desirable properties:

- The client/broker/server model allows a sufficient degree of flexibility in service deployment.
- The architecture is able to provide mathematical services in a platform-neutral manner by making appropriate use of certain accepted XML technologies, including OWL, OpenMath and MathML.
- The framework allows rich problem descriptions. This allows servers to publish their capabilities making use of detailed mathematical ontologies, clients to specify the properties of problem they wish to solve, and brokers to either match and forward service requests, or to restructure requests in a sophisticated manner.
- Mathematical services can be defined by domain experts, requiring no knowledge of web technologies.

The framework allows rich problem descriptions. This allows servers to publish their capabilities making use of detailed mathematical ontologies, clients to specify the properties of problem they wish to solve, and brokers to either match and forward service requests, or to restructure requests in a sophisticated manner.

Mathematical services can be defined by domain experts, requiring no knowledge of web technologies.

Additional XML data formats were straightforward to define and use as needed. These included a Mathematical Services Description Language, a Mathematical Problem Description Language, a Mathematical Explanation Language and a Mathematical Services Configuration Language. The problem of conversion between platform-neutral and platform-specific formats (specifically, between OpenMath and Maple) was solved in a general, extensible fashion using XML configuration modules.

The near-universal support for XML in software packages has made it straightforward to exchange the richly structured objects needed for mathematical computing. Not only has this accelerated the development of our research prototype, it also provides an acceptable external interface for our mathematical web services architecture.

Acknowledgements

The authors would like to thank all their collaborators in the MONET project. Special thanks go to Clare So for her help with the OpenMath to Maple transformation, to Olga Caprotti and Daniele Turi for their contributions to the

Monet ontologies, to Juliette Mainka for the design and implementation of the Mathematical Object Manager, and to Yannis Chicha, Manfred Riem, Stephen Buswell and David Roberts for their contributions to the implementation of the broker.

Bibliography

[AHM05] Simone A. Ludwig, William Naylor, Julian Padget and Omer F. Rana, *"Matchmaking Support for Mathematical Web Services"*. In Proceedings of 4th UK e-Science Programme All Hands Meeting (AHM) 2005, Nottingham, UK.

[DAMLOIL] DAML+OIL, Ian Horrocks, Frank van Harmelen, Peter Patel-Schneider, Tim Berners-Lee, Dan Brickley, Dan Connolly, Mike Dean, Stefan Decker, Dieter Fensel, Richard Fikes, Pat Hayes, Jeff Heflin, Jim Hendler, Ora Lassila, Deb McGuinness , Lynn Andrea Stein, *Technical Report* REC-xslt-19991116, Joint US/EU ad hoc Agent Markup Language Committee, March 2001. http://www.daml.org/2001/03/daml+oil-index.html

[EW02] Mike Dewar & David Carlisle *"Description Schemes For Mathematical Web Services"*. In Proceedings of EuroWeb 2002 Conference, Oxford, UK. http://ewic.bcs.org/conferences/2002/euroweb/session3/paper2.pdf

[IAMC02] Stephen M. Watt, *"Exploiting implicit mathematical semantics in conversion between TeX and MathML,"* In Proceedings of Internet Accessible mathematical Communication (IAMC 2002), July 2002, Lille, France. http://www.symbolicnet.org/conferences/iamc02

[IAMC04] Marc Laurent Aird, Walter Barbera Medina, James Davenport, Julian Padget, *"Description and generation of mathematical web services"*, In e-Proceedings IAMC 200, Santander, Spain. http://www.orcca.on.ca/conferences/iamc2004/papers/MWS_DescriptionAndGeneration.pdf

[ICMS02] Olga Caprotti and Wolfgang Schreiner. *"Towards a Mathematical Services Description Language"*. In Proceedings of ICMS 2002, pp 258 - 265, International Congress of Mathematical Software, Beijing, China, 2002.

[IS] iS - Instance Store, 2005 http://instancestore.man.ac.uk/

[Maple] Maple User Manual, Maplesoft, a division of Waterloo Maple Inc., 2005.

[MathBroker] MathBroker - A Framework for Brokering Distributed Mathematical Services, 2005 http://poseidon.risc.uni-linz.ac.at:8080/mathbroker/index.xml

[MathML] *"Mathematical Markup Language (MathML) Version 2.0 (Second Edition)*, R. Ausbrooks, S. Buswell, D. Carlisle, S. Dalmas, S. Devitt, A. Diaz, M. Froumentin, R. Hunter, P. Ion, M. Kohlhase, R. Miner, N. Poppelier, B. Smith, N. Soiffer, R. Sutor, S. Watt, World Wide Web Consortium Recommendation 21 October 2003. http://www.w3.org/TR/2003/REC-MathML2-20031021

[MathML02] Sandy Huerter, Igor Rodionov, and Stephen Watt, "Content-Faithful Transformations for MathML", In Proceedings of MathML 2002, Chicago, Illinois. http://www.mathmlconference.org/2002/presentations/huerter/

[MKM04a] Elena Smirnova, Clare So and Stephen M. Watt, *"An Architecture for Distributed Mathematical Web Services"* In Proceedings of MKM 2004, pp 363-377, LNCS 3119 Springer-Verlag 2004.

[MKM04b] Olga Caprotti, Mike Dewar and Daniele Turi. *"Mathematical Service Matching Using Description Logic and OWL"*. In Proceedings of MKM 2004, pp 73-87, LNCS 3119, Springer-Verlag 2004.

[NAG] NAG Numerical Components, 2005 http://www.nag.co.uk/numeric/numerical_libraries.asp

[NAMKM04] Elena Smirnova, Stephen M. Watt, *"An Approach to Mathematical Notation Selection"*, In Abstracts Proceedings of NA MKM 2004, Phoenix, Arizona, US. http://imps.mcmaster.ca/na-mkm-2004/proceedings/pdfs/smirnova-abstract.pdf

[OM1] S. Dalmas, M. Gaetano and S.M. Watt, *An OpenMath v1.0 Implementation*, pp. 241-248, Proc. International Symposium on Symbolic and Algebraic Computation, (ISSAC 1997), July 21-23 1997, Kihei, Hawaii, USA, ACM Press 1997. http://www.csd.uwo.ca/~watt/pub/reprints/1997-issac-openmath.pdf

[OM2] The OpenMath Standard 2.0. S.Buswell, O.Caprotti, D.P.Carlisle, M.C.Dewar, M.Gaetano and M.Kohlhase, 2004 http://www.openmath.org/cocoon/openmath/standard/om20/index.html

[ORCCA00] I. Rodionov and S.M. Watt, Ontario Research Centre for Computer Algebra, University of Western Ontario, Research Report TR-00-14, 2000. http://www.orcca.on.ca/TechReports/2000/TR-00-24.html

[OWL] OWL, Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L., McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference. Technical Report REC-owl-ref-20040210, The Worldwide Web Consortium, February 2004. http://www.w3.org/TR/2004/REC-owl-ref-20040210/

[Protege] Protege, 2005 http://protege.stanford.edu/plugins/owl

[TUG02] Stephen M. Watt, *"Conserving implicit semantics in conversion between TeX and MathML"*, TUGBoat, Vol 23, No 1, 2002.

[WSDL] Web Services Description Language (WSDL), Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, Worldwide Web Consortium Note, March 2002. http://www.w3.org/TR/wsdl.html

[WSJ] Mike Dewar, *"Identifying and Brokering Mathematical Web Services"*. Web Services Journal, July 2003. http://webservices.sys-con.com/read/39833.htm

[XSD] XML Schema Part 1: Structures Second Edition, Henry S. Thompson, David Beech, Murray Maloney,Noah Mendelsohn, World Wide Web Consortium Recommendation 28 October 2004. http://www.w3.org/XML/Schema

[XSLT] XSL Transformations (XSLT), James Clark, World Wide Web Consortium Recommendation 16, December 1999. http://www.w3.org/TR/XSLT

Biography

Mike **Dewar**
Numerical Algorithms Group [http://www.nag.com/]
Wilkinson House, Jordan Hill Road
Oxford
United Kingdom
mailto:miked@nag.co.uk

Dr. Mike Dewar is the Vice-President for Research & Development at the Numerical Algorithms Group Ltd. Dr. Dewar has coordinated four major projects funded by the European Union and is also the Vice-President of the OpenMath Society [http://www.openmath.org]. A common thread running through all these activities is a desire to make mathematical computations accessible via the Internet, and to automate and simplify the process of using mathematical software as much as possible.

Elena **Smirnova**
University of Western Ontario [http://www.uwo.ca/]
Ontario Research Centre for Computer Algebra [http://www.orcca.on.ca/]
Department of Computer Science, University of Western Ontario
London
Ontario
N6A 5B7
Canada
mailto:elena@orcca.on.ca

Dr. Elena Smirnova is a research scientist at the Ontario Research Centre for Computer Algebra at the University of Western Ontario. Her research interests lie in the area of mathematical data communication, pen-based mathematical computing and mathematical web services. She holds a joint PhD in Mathematics and Computer Science from the State University of St. Petersburg, Russia and from the University of Paris 12, Val de Marne, France.

Stephen M. **Watt**
University of Western Ontario [http://www.uwo.ca/]
Ontario Research Centre for Computer Algebra [http://www.orcca.on.ca/]
Department of Computer Science, University of Western Ontario
London
Ontario
N6A 5B7
Canada
mailto:watt@uwo.ca

Dr. Stephen Watt is a professor of Computer Science at the University of Western Ontario, where he directs the Ontario Research Centre for Computer Algebra. Prior to joining the University of Western Ontario, Dr. Watt was a research scientist at the IBM T.J. Watson Research Center and a professor at the University of Nice. Dr. Watt is one of the original authors of the Maple computer algebra system and the MathML standard for mathematical markup.